10:49:20 OCA PAD INITIATION - PROJECT HEADER INFORMATION 01/21/88

Active

Project #: G-36-632
Center # : R6439-0A0
Cost share #: G-36-371
Center shr #: F6439-0A0
Rev #: 0
OCA file #:
Work type : RES
Document : GRANT
Contract entity: GTRC

Contract#: NCR-8704850
Prime #:
Mod #:

Subprojects ? : N
Main project #:

Project unit: ICS
Unit code: 02.010.142
Project director(s):
AMMAR M H
ICS

Sponsor/division names: NATL SCIENCE FOUNDATION / GENERAL
Sponsor/division codes: 107 / 000

Award period: 871201 to 890531 (performance) 890831 (reports)

Sponsor amount	New this change	Total to date
Contract value	44,085.00	44,085.00
Funded	44,085.00	44,085.00
Cost sharing amount		10,640.00

Does subcontracting plan apply ?: N

Title: PERFORMANCE ANALYSIS OF LARGE SCALE INFORMATION SYSTEMS

PROJECT ADMINISTRATION DATA

OCA contact: John B. Schonk 894-4820

Sponsor technical contact

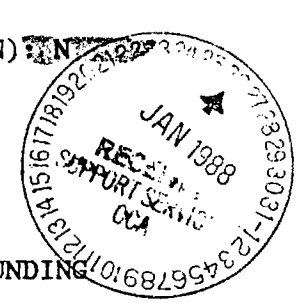
DARLEEN FISHER
(202)357-9717
NATIONAL SCIENCE FOUNDATION
ENG/NCR
WASHINGTON, DC 20550

Sponsor issuing office

STANLEY C. DOBSON
(202)357-9630
NATIONAL SCIENCE FOUNDATION
DGC/ENG
WASHINGTON, DC 20550

Security class (U,C,S,TS): U
Defense priority rating :
Equipment title vests with: Sponsor

ONR resident rep. is ACO (Y/N): N
supplemental sheet
GIT X



Administrative comments -
PROJECT INITIATION
IT IS ANTICIPATED THAT THERE WILL BE 1 ADDITIONAL YEAR OF PROJECT FUNDING

GEORGIA INSTITUTE OF TECHNOLOGY
OFFICE OF CONTRACT ADMINISTRATION

NOTICE OF PROJECT CLOSEOUT

Closeout Notice Date 03/11/91

Project No. C-36-632 Center No. R6439-0A0

Project Director AMMAR M H School/Lab COMPUTING

Sponsor NATL SCIENCE FOUNDATION/GENERAL

Contract/Grant No. NCR-8704850 Contract Entity GTRC

Prime Contract No.

Title PERFORMANCE ANALYSIS OF LARGE SCALE INFORMATION SYSTEMS

Effective Completion Date 901130 (Performance) 910228 (Reports)

Closeout Actions Required:	Y/N	Date Submitted
Final Invoice or Copy of Final Invoice	N	
Final Report of Inventions and/or Subcontracts	Y	910304
Government Property Inventory & Related Certificate	Y	
Classified Material Certificate	N	
Release and Assignment	N	
Other	N	

Comments NO INVOICE REQUIREMENT-NSF LINE OF CREDIT. PATENT REPORT REQUIREMENT SATISFIED BY 98A.

Subproject Under Main Project No.

Continues Project No.

Distribution Required:

Project Director	Y
Administrative Network Representative	Y
GTRI Accounting/Grants and Contracts	Y
Procurement/Supply Services	Y
Research Property Managment	Y
Research Security Services	N
Reports Coordinator (OCA)	Y
GTRC	Y
Project File	Y
Other	N
	N

PERFORMANCE ANALYSIS OF LARGE SCALE INFORMATION SYSTEMS

Progress Report for NSF Grant NCR-8604850
Covers Period Until August 31, 1988

Mostafa H. Ammar

School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332

1. Introduction

This progress report discusses work that was carried out with the funding provided by NSF under grant number NCR-8604850. The description to follow is subdivided into three parts. The first part discusses work relating to analysis of the use of the broadcast delivery mechanism in large scale information systems. The next two sections discuss issues that may arise in a large scale information system with multiple databases:

1- The use of broadcast/multicast communication to determine the location of an appropriate database to search. (This is addressed in the context of the general resource finding problem in distributed systems.)

2- The use of voting to maintain the consistency of a distributed database system.

We briefly describe our research efforts next. A more detailed treatment is included in the set of accompanying technical reports.

The NSF grant is currently supporting a Ph.D student, John Guthrie. Mr. Guthrie started working on the projext in April 1988. He has passed his Ph.D. general exams in June 1988 and is currently preparing a Ph.D. Thesis Proposal. The work described in Sections 2.2 and 2.3 will constitute the "research reults" part of the proposal, which will be completed by November 1988.

2. The Use of Broadcast Communication for Information Delivery

In this section we discuss three studies of the use of broadcast communication to provide the performance enhancing shared response feature. The first study has been completed and a technical report (currently in draft form) has been issued. (The report is attached.) The report is currently being revised for journal submission. The next two studies are still underway and only preliminary results have been obtained.

2.1. Scheduling algorithms for broadcast information delivery systems

An important operational strategy of information systems using broadcast delivery is the scheduling algorithm used to select the next request for processing. In [1,2] a first-come, first-served (FCFS)

discipline is considered, with the modification that a new request joins the queue at the same position as a request for the same page, if such a request is already in the system. Analytic results for this discipline are available for the case of Poisson arrivals and exponentially distributed request processing times.

In this study we formulate the scheduling problem as a Markov decision process. The scheduling decisions and the request arrival process determine the state transitions in our model. An algorithm developed by Howard [3] is used to determine the scheduling decisions that minimize the mean response time over all requests in the system. Due to the complexity of the algorithm it is not possible to obtain the optimal scheduling decisions for systems with a realistic number of users and pages. Thus, we first obtain the solutions for several small example systems using Howard's algorithm. Using the properties of the optimal scheduling decisions indicated by these solutions we developed three scheduling heuristics which are described and evaluated (using simulation) in the attached technical report.

2.2. Protocols for information delivery over a VSAT network

Very small aperture terminal (VSAT) networks possess a star topology in which a central computer (hub) communicates with a large number of remote terminals over a satellite channel [4,5]. (See Figure 1.) This is an ideal architecture for the provision of a broadcast delivery information service. Our ultimate objective in this study is the development of an efficient information delivery protocol for use over a VSAT network. Before such a protocol is developed, however, we need to understand the effect of the considerable delay introduced in the up- and down-links on the response time performance of broadcast delivery. To that end, we will consider the model shown in Figure 2 and the following ways of configuring the system:

Model 1: *No-Requests-Discarded*

Decision 1 = Null (i.e., always admit requests)

Decision 2 = Null

Model 2: *Superfluous-Requests-Discarded*

Decision 1 = Discard request if similar request is in the Hub

Decision 2 = Null

Model 3: *Improved superfluous-Requests-Discarded*

Decision 1 = Discard request if similar request is in the Hub or in the downlink

Decision 2 = Null

Model 4: *Ideal (Not Achievable)*

Decision 1 = Null

Decision 2 = Discard requests (at the terminals) if similar request is in the uplink, Hub or downlink.

The reason for discarding requests in a broadcast delivery system is to avoid superfluous transmissions, i.e., those that do not satisfy any requests. The only configuration that guarantees no superfluous transmissions is that shown in Model 4. This configuration has the additional advantage of reducing the request traffic and thus reducing uplink contention. Unfortunately such a strategy is not feasible as it requires instantaneous state knowledge by the user-terminals.

Our work involves exact and approximate analysis of all four models to understand the response time behaviour of the different configurations under varying request traffic conditions. As a result of such an analysis a set of realizable efficient protocols will be proposed. The work is nearing completion and a technical report discussing the results is currently in preparation.

2.3. Broadcast delivery systems with multiple page requests

The specific problem discussed here is motivated by the work performed at Bellcore on a high throughput database system called *the datacycle architecture* [6]. In that system, high transaction throughput is achieved by broadcasting all the contents of the database continuously over a high bandwidth transmission medium. Read transactions are processed locally by filtering the information received to achieve the desired response. A separate update channel is provided to handle write transactions. The work at Bellcore has consisted of the design and prototyping of hardware filters capable of operating at the high data rates involved (~4-5 Gbps) and the development of commit protocols.

Our work abstracts the above system in order to understand the issues involved in optimizing read

query response time. (Discussions with the people at Bellcore has indicated that they are interested in such results.) In our model the database is subdivided into a set of N discrete units which we call pages. Each read query is analyzed locally to obtain a set, Ω, of required pages. These pages are then fetched from the broadcast stream. Processing of the pages to produce the desired response, e.g., joining of files, is performed locally. In previous work [7,8,9] we have considered the special case when $|\Omega| = 1$.

Our preliminary results are derived based on the availability of a probability distribution q_Ω for $\Omega \subseteq \{1,2,\cdots,N\}$ such that $\sum_{\Omega} q_\Omega = 1$. An expression for the mean response time of such a system has been developed. Based on this expression some preliminary results regarding the optimization of the response time have been obtained. It is expected that our work will lead to some concrete proposals for scheduling transmissions in the datacycle architecture.

3. Using Broadcast/Multicast Communication to Find Resources in a Distributed System

In any distributed system, it is necessary to implement procedures to find resources when only the resource name or sometimes property is known. This allows the users of the system to see the abstraction of a unified system in which the resource's location is transparent. One widely used scheme is the broadcasting of a resource finding request to all nodes. This approach is simple and requires nodes to know only about their local resources. Broadcasting is, however, expensive since all system nodes are interrupted and asked to search local resource directories every time a find request is made. In the work described here multicast communication is used to implement resource finding procedures. This helps reduce the number of nodes involved in every resource finding request at the expense of procedure complexity or response time.

This work is related to the work described in Section 2 in two ways:

1- It considers the use of broadcast/multicast communication; the heart of our large scale information system studies; and

2- It may be applied to the problem of locating an appropriate database in an information system with multiple databases.

The work summarized below has resulted in two technical reports which are attached. One will appear in the proceedings of the 13th IEEE Conference on Local Computer Networks. The other has been submitted to the IEEE INFOCOM '89 conference. Both are currently being revised for journal submission.

3.1. Using multicast communication to locate resources in a LAN-based distributed System

In the multicast scheme developed in this work, the universe of resource names is partitioned into a relatively small number of groups and each group is assigned a unique address. Nodes storing the locations of resources belonging to a particular group instruct their network interfaces to receive all location messages sent to the group address. To locate a resource, a node first determines the address of the group to which the resource belongs (this can be accomplished via a well known hash function), and a multicast message is then sent to the address. We also developed algorithms to be executed when a resource is created or deleted.

The algorithm performance is studied by means of simulation and closed form solutions are derived for systems operating at heavy and light loads. The cost measure used is the number of nodes that process messages sent for finding a resource or for updating the information stored by nodes when resources are added or deleted. The scheme's performance is compared with that of broadcast, and it is shown that the proposed scheme performs much better than broadcast alone.

3.2. Optimal selection of multicast groups for resource location in a distributed system

At the other end of the spectrum from the broadcast location mechanism discussed above is the procedure by which the individual nodes are polled sequentially. This would certainly decrease the amount of CPU time wasted in the system (especially if the nodes more likely to know about the resource were consulted first). However, this approach would also increase the bandwidth utilization, because many messages will be sent. Since the messages are sent sequentially, the real disadvantage of this approach is that the location operations would take longer.

In this work we design a location protocol which considers a cost measure that includes both the CPU utilization and the response time. The approach taken is based on a scheme in which the nodes in

the network are divided into disjoint multicast groups. To find a resource's whereabouts, these groups are polled sequentially with multicast messages. The two approaches mentioned above (broadcast and polling) are just special cases when all nodes are reached by a single message or when only one node is reached with each message.

We develop a cost model for the system based on realistic network assumptions. Using the model, we give an efficient algorithm which finds the optimal decomposition into disjoint groups, as well as the optimal sequence in which the groups should be probed. The algorithm can make use of knowledge of the probability distribution of a resource's whereabouts, if such information is available.

4. Optimizing Vote and Quorum Assignments for Reading and Writing Replicated Data

Voting has been used for various applications in distributed systems. Its use for synchronizing read and write operations on replicated files was first proposed by Gifford [10]. Each copy is assigned some number of votes and each operation is required to obtain a pre-defined quorum to proceed. To ensure that a read operation returns the value installed by the last write operation, the read and write operations must acquire r and w number of votes respectively such that $r + w > L$, where L is the total number of votes assigned to all copies. The values r and w are called the read and write quorums.

In previous work [11], we evaluated the use of the voting mechanism to manage read and write transactions. For systems where node reliabilities are identical, values are derived for the optimal degree of replication and for the optimal read and write quorums. In the work described in the attached technical report, we study the problem of finding the vote and quorum assignments that result in the best system availability for read and write operations in a system where node reliabilities may be different. For this problem, the proportion of read and write operations have to be taken into account. Although the number of vote assignments is theoretically unbounded, the set of vote assignments that should be considered for best performance for reading and writing is finite. We first discuss an algorithm that generates vote assignments. The algorithm is based on a fundamental relationship between systems with N and systems with $N+1$ nodes. The actual vote and quorum assignments are obtained as solutions to a Linear Program. A method is developed to compute the availability of the system with a given vote assignment. By applying the method to the enumerated set of vote assignments the optimal one is

determined.

The paper describing this work has been accepted to the IEEE 5th Data Engineering Conference and is being revised for Journal submission.

REFERENCES

[1] Wong, J. W., Ammar, M. H., "Response Time Performance of Videotex Systems," *IEEE J. on Selected Areas in Communications,* Vol. 4, # 7, November 1986.

[2] Wong, J. W., Ammar, M. H., "Analysis of Broadcast Delivery in a Videotex System," *IEEE Transactions on Computers,* Vol. 34, # 9, September 1986.

[3] Howard, R. A., *Dynamic Programming and Markov Processes,* MIT Press, Cambridge, MA, 1960.

[4] Chitre, D. M ., McCoskey, J. S., "VSAT Networks: Architectures, Protocols and Mangement," *IEEE Communications Magazine,* Vol. 26, # 7, July 1988.

[5] Stratigos, J., Mahindru, R., "Packet Switch Architectures and User Protocol Interfaces for VSAT Networks," *IEEE Communications Magazine,* Vol. 26, # 7, July 1988.

[6] Herman, G., Gopal, G., Lee, K.C., Weinrib, A., "The Datacycle Architecture for Very High Throughput Database Systems," Proc. of ACM SIGMOD International Conference on Management of Data, San Fransisco, 1987.

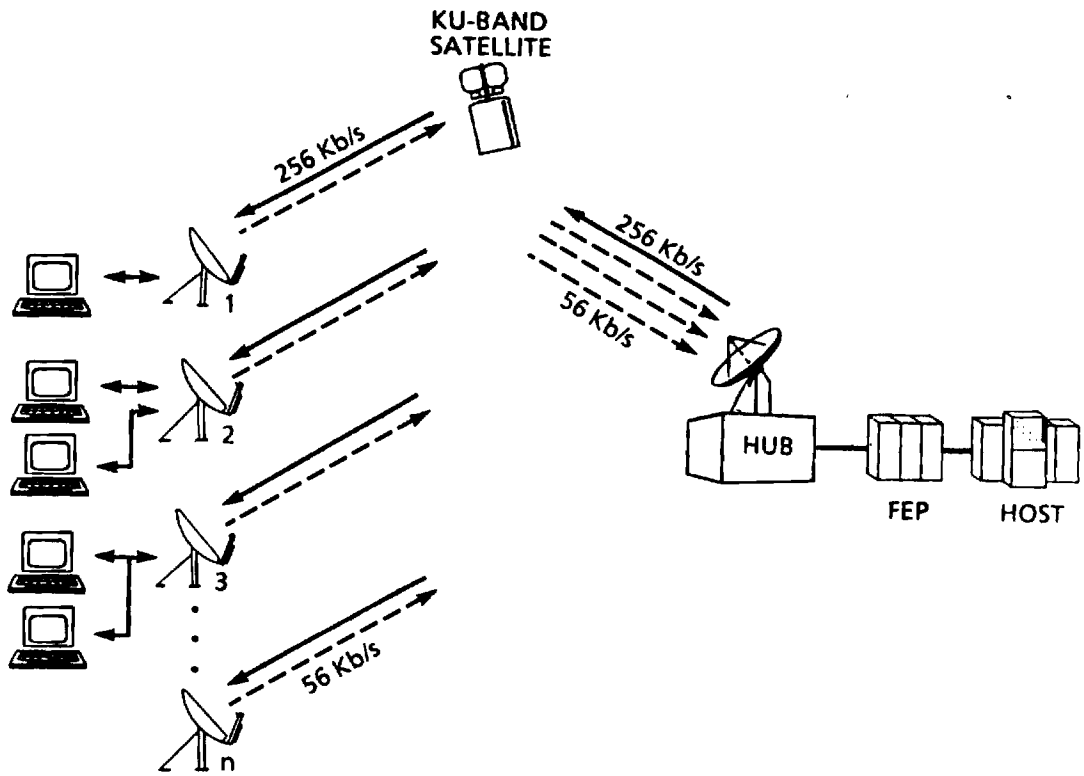
[7] Ammar, M. H., Wong, J. W., "The Design of Teletext Broadcast Cycles," *Performance Evaluation,* Vol. 5, # 4, November 1985.

[8] Ammar, M. H., Wong, J. W. , "On the Optimality of Cyclic Transmission in Teletext Systems," *IEEE Transactions on Communications,* Vol. 35, # 1, January 1987.

[9] Ammar, M. H., "Response Time in a Teletext System: An Individual User's Perspective," *IEEE Transactions on Communications,* Vol. 35, # 11, November 1987.

[10] Gifford, D., "Weighted Voting For Replicated Data," Proceedings of 7th Symposium on Operating Systems, ACM, December 1979.

[11] Ahamad, M., Ammar, M. H., "Performance Characterization of Quorum-Consensus Algorithms for Replicated Data," To appear in *IEEE Transactions on Software Engineering.* (Earlier Version in Proc. of the 6th Symposium on Reliability in Distributed Software and Database Systems.)



KEY:
FEP - FRONT END PROCESSOR

Figure 1 VSAT network configuration

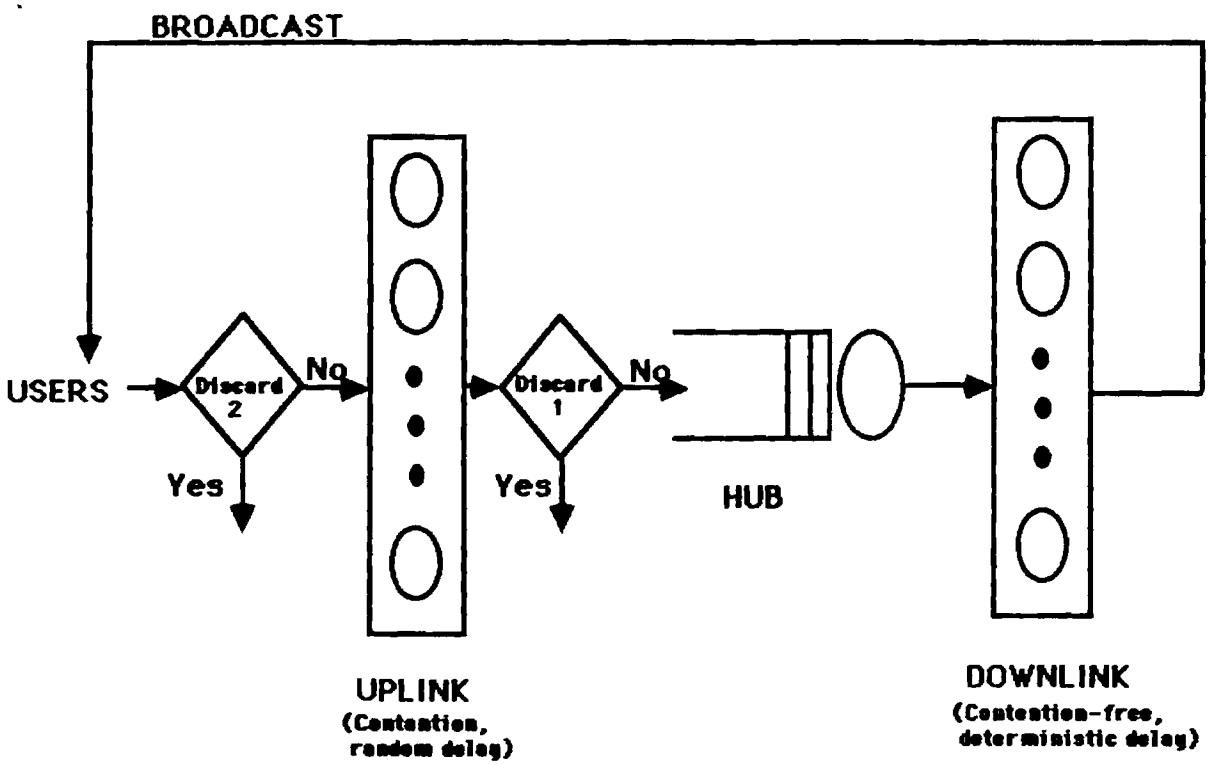


Figure 2 Model for information delivery on a VSAT network

Scheduling Algorithms for Broadcast Information Delivery Systems

H.D. Dykeman and J.W. Wong

Department of Computer Science

University of Waterloo

Waterloo, Ontario, Canada N2L 3G1

Tel. (519) 888-4431

M.H. Ammar

School of Information and Computer Science

Georgia Institute of Technology

Atlanta, GA 30332

Draft 4.

Abstract

We consider an interactive information delivery system in which user requests are forwarded to a service computer, which processes the requests and returns the desired information to the users. If a broadcast transmission medium is available, the information can be broadcast to all users. The use of broadcast delivery satisfies all pending requests for the same information, resulting in an improvement in the response time performance of the system. In this paper a simple model of a broadcast information delivery system is used in order to examine scheduling algorithms for selecting the next request to be processed. The performance measure of interest is the mean response time over all requests in the system. The scheduling problem is formulated as a Markov decision process, to determine desirable scheduling properties. Scheduling algorithms are developed based on these properties, and their performance is compared using simulation. Implementation issues for the proposed algorithms are also discussed.

1. Introduction

Information delivery systems provide their users with timely access to a variety of information. Applications of such systems can be found in commercial and educational environments, providing access to financial and technical information. With the introduction of the Integrated Services Digital Network, such systems may also become widely available in the home [1,2]. The range and quality of services available to commercial users will also be enhanced due to the availability of high bandwidth communication networks [3].

The major components of the information delivery system under consideration are shown in Figure 1. Information is organized into discrete units called *pages* and stored on disk. Users submit requests, and receive the requested pages, through user terminals. Requests are forwarded via a communication network to a service computer. Based on the requests that are pending, the service computer retrieves pages from disk and delivers them via the network to the users. Videotex is an example of an information delivery system with this architecture [4].

In [5], two methods of page delivery were considered: *individual delivery* where the retrieved page is transmitted to the requesting user only, and *broadcast delivery* where the retrieved page is broadcast to all users. It was shown that the use of broadcast delivery leads to a lower mean response time, and enables the system to handle a higher traffic intensity, since all pending requests for a page are satisfied by the next transmission of that page [5]. Individual delivery is the only alternative if a broadcast transmission medium is not available. It is also required if the requested information is of a confidential nature.

This paper is concerned with the performance of information delivery systems using broadcast delivery. An important operational strategy of such systems is the scheduling algorithm used to select the next request for processing. In [5,6] a first-come first-served (FCFS) discipline is considered, with the modification that a new request joins the queue at the same position as a request for the same page, if such a request is already in the system. Analytic results for this discipline are available in [5,6] for the case of Poisson arrivals and exponentially distributed request processing times.

In this paper we formulate the scheduling problem as a Markov decision process. The scheduling decisions and the request arrival process determine the state transitions in our model. An algorithm developed by Howard [7] is used to determine the scheduling decisions that minimize the mean response time over all requests in the system. Due to the complexity of the algorithm it is not possible to obtain the optimal scheduling decisions for systems with a realistic number of users and pages. Our strategy is to obtain the solutions for several small example systems and then develop scheduling algorithms using the properties of optimal scheduling decisions indicated by these solutions. We compare the performance of our proposed scheduling algorithms to FCFS using a simulation model.

This paper is organized as follows. In section 2 our performance model for broadcast information delivery systems is described. The Markov decision process formulation used to obtain the optimal scheduling decisions is presented in section 3. Several examples are solved in section 4, and some useful properties of the optimal scheduling decisions are identified. These properties are then used to develop new scheduling algorithms, which are presented in section 5. We examine the response time performance of these scheduling algorithms in section 6 and discuss implementation issues in section 7. Our results are summarized in section 8.

2. Model Description

Our model of a broadcast information delivery system is shown in Figure 2. The system has K users whose think times are assumed to be independent and exponentially distributed with mean $1/\gamma$. There are N information pages, and the probability that a request is for page i is assumed to be q_i, $i = 1, 2, \cdots, N$, where $\sum_{i=1}^{N} q_i = 1$. The request processing times (to retrieve and broadcast a page) are assumed to be independent and exponentially distributed with mean $1/\mu$.

We denote the state of our model by $[(n_1, n_2, \cdots, n_N), j]$ where n_i is the number of pending requests for page i, $0 \leq n_i \leq K$, and $n = \sum_{i=1}^{N} n_i \leq K$. The parameter j represents the

state of the service computer. When $j = 0$ the service computer is idle, and for $j = 1, 2, \cdots, N$, it is processing a request for page j. We only consider scheduling policies for which $j > 0$ when $n \geq 1$ (i.e., the server will never be idle if there is at least one request pending).

State transitions in our model are caused by request arrivals as well as service completions (see Figure 3). Since think times and service times are exponentially distributed, our model is Markovian. When an arrival of a request for page i occurs, a system in state $[(n_1, \cdots, n_i, \cdots, n_N), j]$ will enter state $[(n_1, \cdots, n_i{+}1, \cdots, n_N), j]$. The transition rate is given by $q_i(K - n)\gamma$. Transitions out of a state due to a service completion occur at rate μ and are only possible when $j > 0$. Consider the transition out of state $[(n_1, \cdots, n_j, \cdots, n_N), j]$ due to a service completion. The next state entered is $[(n_1, \cdots, n_j = 0, \cdots, n_N), j_2]$ ($n_j = 0$ because the broadcast of page j satisfies all pending requests for that page). Implicit in this transition is that the system makes a decision to process the requests for page j_2 next. We assume that this decision is made according to the following rule:

$j_2 = 0$ *if* $n = 0$,

otherwise $j_2 \in \{ m \mid n_m \geq 1, m{=}1, 2, \cdots, N \}$.

In other words, the decision is made among pages that have at least one pending request.

To completely define the Markovian state transitions associated with our model, we need to establish a *next-page-to-process* decision for each state. The decision for a particular state is denoted by $d[(n_1,n_2, \cdots, n_N), j]$. Setting $d[(n_1,n_2, \cdots, n_N), j] = j_2$ implies that if the processing of page j completes while the system is in this state then the next page to be serviced is page j_2. The set of decisions for all feasible states defines a *scheduling policy*.

3. Markov Decision Process Formulation

Our objective is to obtain a scheduling policy for the model described above such that the mean response time is minimized. Response time under broadcast delivery is defined to be the elapsed time from when a request is submitted until the completion of the next broadcast of the

requested page. For convenience we assume that a request will be satisfied at the end of the current transmission, if it arrives during the broadcast of the requested page. Our approach is to formulate the scheduling problem as a Markov decision process and relate the cost function to response time. There are two ways in which a Markov decision process incurs costs [7]:

(i) *transition cost* which is incurred in a lump sum when a state transition occurs, and

(ii) *state occupancy cost* which is directly proportional to the time spent in each state.

For our model the transition cost is assumed to be 0. Occupancy of state $[(n_1,n_2,\cdots,n_N), j]$ incurs cost at a rate of n per unit time, where $n = \sum_{i=1}^{N} n_i$. For a given scheduling policy, let $C(t)$ be the instantaneous rate at which cost is being incurred at time t. We have

$$C(t) = n(t) \tag{1}$$

where $n(t)$ is defined to be the total number of pending requests at time t. If $V(T)$ is the total cost incurred up to time T, then for a given scheduling policy we have

$$V(T) = \int_0^T C(t)dt. \tag{2}$$

We define the average cost per unit time of using the given scheduling policy as

$$\eta = \lim_{T\to\infty} \frac{1}{T} V(T) = \lim_{T\to\infty} \frac{1}{T} \int_0^T C(t)dt. \tag{3}$$

Note that the limit in (3) exists for our model since the cost function has a maximum value equal to the number of users K (i.e., $0 \leq \eta \leq K$). Note also that $C(t)$ represents the number of requests in the system at time t and thus η represents the mean number of requests in the system using the given policy. This is directly related to the mean response time through Little's result [8]. Therefore minimizing the average cost per unit time of operating the system will also minimize its mean response time.

Howard [7] has developed a *policy-iteration* algorithm, which can be used to obtain a scheduling policy that minimizes η for our model. Initially an arbitrary scheduling policy is specified from which all state transition rates are determined. The general form of state transitions for our model was shown in Figure 3. We use a_{rs} to denote the transition rate from state r to s. Also, for each state we specify C_r, the cost per unit time of remaining in state r, which is given by $\sum_{i=1}^{N} n_i$ if state $r = [(n_1, n_2, \cdots, n_N), j]$.

Let P be the total number of states. The first stage of Howard's policy-iteration algorithm, the *Value-Determination Operation,* uses a_{rs} and C_r to solve the set of equations

$$\eta = C_r + \sum_{s=1}^{P} a_{rs} v_s, \quad r = 1,2, \cdots, P$$

for v_s and η by setting v_P to zero. v_s can be interpreted as a relative measure of the cost of occupying state s, and η as a relative measure of the long term average system cost.

In the second stage of Howard's algorithm, the *Policy-Improvement Routine,* we use the v's obtained from the first stage and change the scheduling policy (and therefore the state transition rates a_{rs}) so as to minimize $C_r + \sum_{s=1}^{P} a_{rs} v_s$ for all states r. The new values for a_{rs} are used in the next iteration of the algorithm. The two stages are repeated until the scheduling policy remains unchanged for successive iterations. At this point the algorithm has converged and the scheduling policy is optimal with respect to minimizing η. Note that Howard's algorithm is guaranteed to converge [7].

4. Properties of the Optimal Scheduling Policy

An immediate difficulty in applying Howard's policy-iteration algorithm to our model is that the complexity of this algorithm is directly proportional to the number of states, which grows rapidly with N and K. Since the system under consideration is providing a variety of information to a large user population, N and K are typically large. We are therefore unable to obtain the optimal scheduling policies when the parameters are set to realistic values.

Our approach is to apply Howard's algorithm to examples with small values of N and K, and identify properties of the optimal scheduling policies. These properties will then be used to develop new scheduling algorithms for selecting the next page to process.

Our examples are based on a small system with 12 users and 3 pages. Consider first the case of equal page request probabilities (i.e., $q_1 = q_2 = q_3 = 0.333$). The mean think time $\frac{1}{\gamma} = 12$, and the mean request processing time $\frac{1}{\mu} = 1$. The optimal scheduling decisions for states in which page 2 is being broadcast (i.e., $d[(n_1,n_2,n_3), 2]$) are presented in Figure 4. We observe that the next page to process is page 1 if $n_1 > n_3$ and vice versa. If $n_1 = n_3 > 0$ at service completion, the tie is broken by arbitrarily choosing page 1 or 3. Similar observations are made for $d[(n_1,n_2,n_3), 1]$ and $d[(n_1,n_2,n_3), 3]$. Other examples for the equal page request probabilities case (results not shown) indicate that the optimal policy is independent of γ and μ. The above observations can be explained as follows. By broadcasting the page with the most pending requests, the short term cost of the system is minimized. This leads to a reduction in the overall mean response time, since the pages have equal request probabilities.

We next consider the following two sets of unequal page request probabilities:

$P1$: $q_1 = 0.5$, $q_2 = 0.3$, $q_3 = 0.2$; and

$P2$: $q_1 = 0.9$, $q_2 = 0.099$, $q_3 = 0.001$.

$P1$ and $P2$ represent situations in which the page request probabilities are slightly different and significantly different respectively. We first consider a system with $\frac{1}{\gamma} = 12$ and $\frac{1}{\mu} = 1$. The optimal scheduling policies for $P1$ and $P2$ are identical, and the scheduling decisions for states in which page 2 is being broadcast are shown in Figure 5. Similar to the equal page request probabilities case, the next page to process is given by the page with the most outstanding requests. However, in case of a tie, the page with the lower request probability is selected. Similar observations are made for states in which page 1 or page 3 is being broadcast. These observations can be explained as follows. When the next-page-to-process decision is any one of the pages involved in a tie, the short term cost of the system is minimized. However, by processing the

requests for the page involved with the lowest request probability first, we increase the chances that the subsequent broadcast(s) of the other page(s) involved will satisfy a greater number of requests due to their higher request arrival rate. This leads to a reduction in the mean response time.

To study the effect of increased load, we reduce the mean think time $1/\gamma$ from 12 to 1.2. Figure 6 shows the optimal scheduling decisions for $d[(n_1,n_2,n_3), 2]$ when the request probabilities are given by $P1$. These results are different from those of Figure 5 (lighter load) in the way ties are broken. Specifically, a *decision threshold* now exists, and the optimal scheduling policy breaks ties in favor of the page with the higher (or lower) request probability if the number of pending requests is below (or above) the threshold. It is important to note that when the system load is heavy, the number of pending requests is likely to be large, and hence the optimal scheduling policy breaks ties in favor of the page with the lower request probability most of the time.

Figure 7 shows the corresponding results for $\frac{1}{\gamma} = 1.2$, with page request probabilities given by $P2$. We observe an increase in the amount of preference towards pages with low probabilities. In some cases, the optimal scheduling policy selects page 3 over page 1 even when $n_3 < n_1$ at the completion of the page 2 broadcast. By not transmitting the page with the most pending requests, the short term cost of the system is increased. However, since pages with low probabilities have long request interarrival times, the current requests for such a page will have long response times, if they are not serviced until additional requests for that page arrive. The cost of having a small number of requests in the system over a long period will eventually exceed the cost of having a larger number of requests over a shorter period. By broadcasting pages with low probabilities when fewer requests are pending, this is avoided. Subsequent broadcasts of pages with higher probabilities will satisfy additional requests due to their higher arrival rate. Thus the long term cost of the system is reduced.

5. Scheduling Algorithms

We now use the results presented in the previous section to develop new scheduling algorithms for determining the next page to process in an information delivery system. Our objective is to design algorithms that make scheduling decisions similar to the optimal policies, in order to achieve good response time performance.

The first scheduling algorithm we propose is referred to as *Most Requests First* (MRF):

"*Select the page with the largest number of pending requests; break ties in favor of the page with the lowest request probability; if more than one page with the largest number of pending requests is tied for the lowest request probability, select one of these pages in an arbitrary manner.*"

Based on the results in the previous section (see Figure 4) we conjecture that MRF is optimal with respect to minimizing the mean response time when the page request probabilities are equal.

The optimal scheduling policies for the case of unequal page request probabilities are much more difficult to characterize. From the results presented in Figures 5, 6, and 7, we observe that the relative priority of a page increases with the number of requests pending for that page, but decreases with its request probability. We therefore propose the following scheduling discipline which we will denote by MRF-B(x) (MRF Biased):

"*Select the page i ($i = 1, \cdots, N$) which maximizes the priority function $F_i = \frac{n_i}{q_i^x}$; break ties in favor of the page with the lowest request probability; if more than one page with the highest priority is tied for the lowest request probability, select one of these pages in an arbitrary manner.*"

The parameter x allows the relative weighting of n_i and q_i in the priority function to be controlled. By increasing x the influence of the page request probabilities is increased, and vice versa. Note that when $x = 0$ or $q_i = q_j$ for all i, j, MRF-B(x) is identical to MRF.

The third scheduling algorithm that we propose makes its decisions based on the waiting time of requests as opposed to the number of requests. The proposed algorithm, *Longest Wait First* (LWF), is defined as follows:

"*Select the page for which the total waiting time of pending requests is largest.*"

This algorithm is analogous to MRF-B(x) in the sense that the priority of a page increases with the number of pending requests. Also, requests for a page with a low request probability will have been waiting longer on average than the same number of requests for a page with a high request probability (due to the longer average interarrival times) and therefore the priority of a page tends to increase as the request probability decreases (relative to MRF).

6. Performance Results

We now compare the mean response time performance of the scheduling algorithms developed in the previous section and of FCFS (modified). Since analytic results are not available for our information delivery system model operating under these scheduling algorithms, we use a simulation model to obtain our performance results. For convenience, we denote the mean response time over all requests in the system by S. We consider an example system with $N = 100$ pages. The mean request processing time is used as our time unit (i.e., $\mu = 1$), and the mean user think time is $\frac{1}{\gamma} = 50$. The page request probabilities are given by $q_i = \frac{c}{i^y}$ $(i = 1, \cdots, N)$, where c is a normalization constant given by $\left[\sum_{i=1}^{N} \frac{1}{i^y}\right]^{-1}$. When the parameter $y = 0$, the request probabilities are constant and equal to $1/N$. As y increases the probabilities become more skewed. At $y = 1$, the request probabilities follow Zipf's law which has been shown to closely approximate user behavior in videotex systems [4].

We first consider the case of equal page request probabilities ($y = 0$, or $q_i = q_j$ for all i, j) with MRF, LWF, and FCFS scheduling (MRF-B(x) is the same as MRF in this case). In Figure 8 we show the mean response time S plotted against the number of users K. When K is small, the load on the system is light and few scheduling decisions are necessary. Therefore S is

insensitive to scheduling when $K < 100$. For $K > 100$, the MRF algorithm exhibits the best mean response time characteristics of the algorithms considered. This further supports our conjecture that the MRF algorithm is optimal for the case of equal page request probabilities.

In Figure 9 we show the probability density function (pdf) for the request response times when $K = 1000$. We observe that under MRF scheduling, the number of requests with very short response times, and the maximum response time are both large relative to the other disciplines. This can be explained as follows. Under MRF, the priority of a page does not depend on the length of time that its requests have been waiting. Therefore a page with a relatively small number of pending requests may not be transmitted until either more requests for that page arrive, or until a large number of other pages are transmitted. This will result in a long delay for some requests. On the other hand, a new request increases the priority of a page immediately. Therefore the arrival of a request may lead to the transmission of the requested page due to the increase in priority, resulting in short delays for some requests. Under LWF scheduling, the number of very long and very short delays is reduced since the scheduling decisions are based on waiting times. FCFS minimizes the maximum response time since the page with the request that has been waiting longest is always the next to be transmitted.

We next consider the case of unequal page request probabilities ($y > 0$) with MRF, MRF-B(x), LWF, and FCFS scheduling. In order to use the MRF-B(x) scheduling algorithm, we must first choose a suitable value for the parameter x. In Figure 10 we show S plotted as a function of x for four different sets of page request probabilities (that are obtained by varying y), for our previous example ($K = 1000$, $N = 100$, $\mu = 1$, $\gamma^{-1} = 50$). We observe that for the case of equal page request probabilities ($y = 0$) the parameter x has no effect, since the priority function F_i is not affected by x. As y increases (i.e., as the request probabilities become more skewed) the value of x resulting in the lowest S decreases. However the best value for x changes very little, even though the request probabilities are changed substantially. We have tuned the parameter x for a relatively high system load ($K = 1000$). Under moderate loads (e.g., $K = 300$) the best values for x are not noticeably affected.

Finally in Figure 10 we observe that as the request probabilities become more skewed, the lowest achievable mean response time decreases significantly. This can be explained as follows. When y is increased, a greater percentage of the requests are for a smaller group of pages. The system can therefore provide good response time to a larger number of requests by transmitting these pages more frequently, resulting in an improvement in the overall mean response time.

In Figure 11 we show S plotted against K for $y = 1$ (i.e., the request probabilities follow Zipf's law). Results for the MRF-B(x) discipline are shown for $x = .5$, the best value from Figure 10. As in the equal page request probabilities case, we observe that when the system is lightly loaded, the response time is insensitive to the scheduling discipline. As K is increased, the MRF-B(.5) algorithm yields the lowest overall mean response time of the algorithms considered. LWF also exhibits good response time characteristics. These results support our observations and conclusions, based on Howard's algorithm, in sections 4 and 5.

The pdf for the request response times when $K = 1000$ in this example is shown in Figure 12. Note that the x-axis is plotted on a log scale. Similar to the equal page requests probabilities case, the use of MRF results in a large number of requests with very short response times and a large maximum response time. MRF-B(.5) reduces these effects compared to MRF since additional priority is given to pages with low request probabilities (i.e., those pages that would normally have a relatively small number of requests in the system). LWF results in a substantial decrease in the maximum response time, while achieving mean response time results comparable to MRF-B(.5). As before, FCFS results in a minimum value for the maximum response time, while the mean respose time is highest.

In Figure 13 we show S plotted as a function of K for $y = 2$ (the remaining parameters are unchanged). As in the previous example, the overall mean response time is lowest under MRF-B(x) (x is tuned to .45 in this case) and LWF scheduling. As y increases the improvement in response time gained by using MRF-B(x) or LWF, instead of MRF or FCFS, becomes more substantial.

7. Implementation Issues

In our information delivery system model there is no representation of scheduling overhead. If this overhead is significant, it may be an important factor in determining which algorithm is most suitable in an implementation. Scheduling functions are performed when a new request arrives at the service computer, and when the next page to process must be determined. In the former case the appropriate data structures must be updated to reflect the additional request, and in the latter case the data structures must be consulted in order to determine the highest priority page, and then updated to reflect the scheduling decision.

It may be possible to update the scheduling data structures in parallel with request processing (i.e., page retrieval and transmission), since this processing is performed chiefly by auxiliary processors (the disk controller and the network adapter). Therefore this overhead may not affect the response time except at high loads, when the arrival rate of requests exceeds the rate at which the data structures can be updated [9].

Determining the next page to process will have a direct impact on the response time of the system since no request processing can be performed until this operation has completed. It is therefore desirable to design the scheduling algorithms so that the next page to process can be determined in constant time. In this section we show that this is possible for all of the scheduling algorithms discussed in this paper.

A FCFS queue of requests can be updated by inserting new requests at the end of the queue in constant time. A simple data structure such as an array can be used in order to detect if a request for a given page is already in the queue. The next page to process can be determined in constant time by removing the element from the head of the queue.

Since priority under MRF depends only on the number of pending requests (and the page request probabilities for MRF-B(x), which we assume are static) we can maintain an ordered (by priority) list of pages that is updated at each request arrival instance. The time complexity of updating the list is $O(N)$, although in most cases only a small number of operations are required (e.g., to follow up a linked list from the current position until the proper new position is located). To obtain the next page to process, the first element can be removed from this list in constant

time.

Under LWF scheduling the priorities vary with time, at a rate dependent on the number of pending requests, and therefore the procedure implementing this scheduling algorithm is more complicated than those for FCFS, MRF, and MRF-B(x). Due to the added complexity of this procedure, we describe an implementation in more detail. In the following description $priority(i)$ is the priority of page i and $update(i)$ is the time at which this priority was last updated. The current time is given by $time$.

Assume that page i is currently the highest priority page in the system (excluding the page being processed). In the absence of request arrivals page i will continue to be the highest priority page for h time units. After this time the priority of some other page, j, will become equal to that of page i. The length of the interval h is given by

$$h = \underset{j=1,\cdots,N;j\neq i}{Min}\left\{\frac{priority(i) - priority(j)}{MAX(n_j - n_i, 0)}\right\}.$$

If page i has at least as many requests as every other page in the system, h is infinite. At the end of the interval h, the priority of each page k is updated as follows:

$$priority(k) = priority(k) + n_k\ (time - update(k))$$

$$update(k) = time.$$

A new value for h is computed relative to the page with the highest priority.

If a request arrives for page k during the interval h, the priority of page k is updated as shown above. If $k \neq i$, the priority of page i is also updated and the system checks to see if the priority of page k will exceed that of page i before the end of the current interval h. If so the length of this interval is reduced accordingly.

When the current request processing completes, the system knows the next page to process by the above algorithm. The priority of this page is set to zero. The priority of all other pages is updated, and a new h is computed relative to the highest priority page.

The time complexity of the above algorithm is $O(N)$ at the end of each interval h, and after the next page to process is determined. A constant number of operations are required at each request arrival instance, and to determine the next page to process.

The time complexities for updating the data structures and for determining the next page to process are summarized in Table 1 for all of the scheduling algorithms considered in this paper. Although the time complexity to update the data structures is the same for MRF, MRF-B(x), and LWF, the frequency with which an update is required is normally greater for LWF, and the actual number of operations required will also be greater in this case.

8. Summary

In this paper we have used a simple model of a broadcast information delivery system in order to develop and compare algorithms for scheduling the processing of requests. The problem was formulated as a Markov decision process and the scheduling policy that minimizes the overall mean response time was obtained for small example systems. Scheduling algorithms were developed based on these optimal policies. These new algorithms, MRF, MRF-B(x), and LWF, were compared with FCFS on the basis of response time performance. The results that we have presented lead to the following conclusions:

(i) FCFS scheduling minimizes the maximum response time. However this benefit is gained at the expense of significantly increasing the mean response time.

(ii) When the page request probabilities are equal, we conjecture that the mean response time of the system will be minimized under MRF scheduling.

(iii) When the page request probabilities are unequal, both MRF-B(x) (with parameter x properly tuned) and LWF have good response time characteristics. LWF has the advantage of substantially reducing the maximum response time. The time complexity for both algorithms is of the same order ($O(N)$ for updates, and $O(1)$ for determining the next page to process), but the actual amount of processing necessary to update the data structures for LWF will usually be greater than for MRF-B(x). If these updates can not be performed in

parallel with request processing, then MRF-B(x) may be preferable to LWF.

Acknowledgement

D. Dykeman and J. Wong were supported by the Natural Sciences and Engineering Research Council of Canada. M. Ammar was supported by the National Science Foundation under grant No. NCR-8604850.

References

[1] Rothamel, H., "ISDN Broadband Services and Applications," Siemens, Telecom Report 9 (1986) No. 1.

[2] Duc, N., "ISDN Terminals and Integrated Services Delivery," IEEE Journal on Selected Areas in Communications SAC-4 (8), Nov. 1986, 1188-1192.

[3] Sugimoto, M., M. Taniguchi, S. Yokoi, and H. Hata, "Videotex: Advancing to Higher Bandwidth," IEEE Communications Magazine 26 (2), Feb. 1988, 22-30.

[4] Gecsei, J., *The Architecture of Videotex Systems,* Prentice-Hall, 1983.

[5] Wong, J.W., and M.H. Ammar, "Response Time Performance of Videotex Systems," IEEE Journal on Selected Areas of Communications SAC-4 (7), Oct. 1986, 1174-1180.

[6] Wong, J. W., M. H. Ammar, "Analysis of Broadcast Delivery in a Videotex System," IEEE Transactions on Computers C-34 (9), Sept. 1985, 863-866.

[7] Howard, R. A., *Dynamic Programming and Markov Processes,* M.I.T. Press, Cambridge, 1960.

[8] Little, J. D., "A Proof of the Queueing Formula $L = \lambda W$," Operations Research 9 (3), May 1961, 383-387.

[9] Wong, J.W., H.D. Dykeman, "Architecture and Performance of Large Scale Information Delivery Networks," to appear in Proc. 12th International Teletraffic Congress, Torino, Italy, June 1-8, 1988.

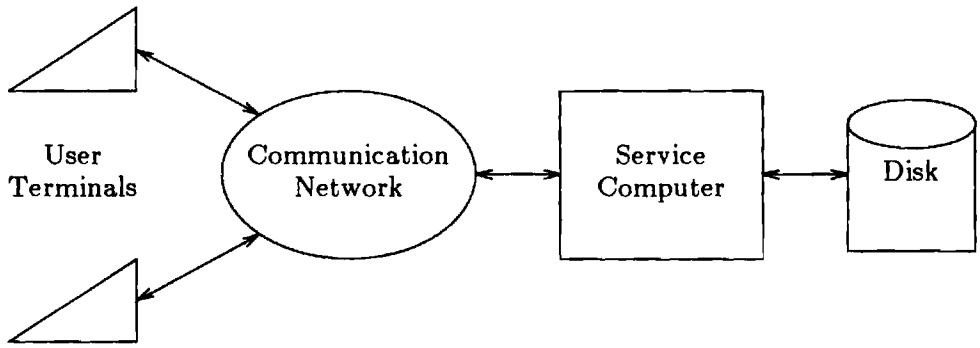


Figure 1. Components of an Information Delivery System.

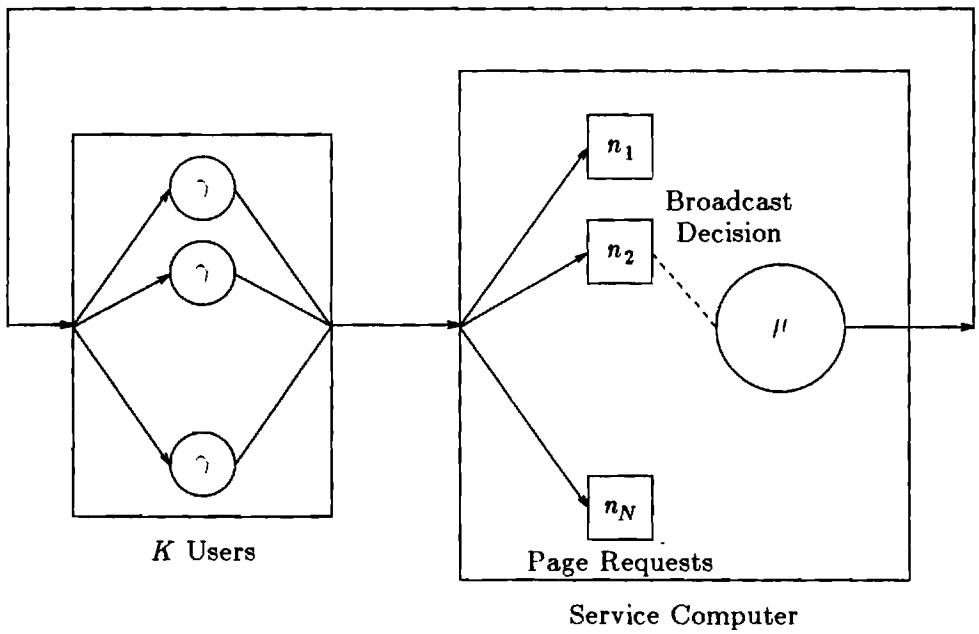


Figure 2. Information Delivery System Model.

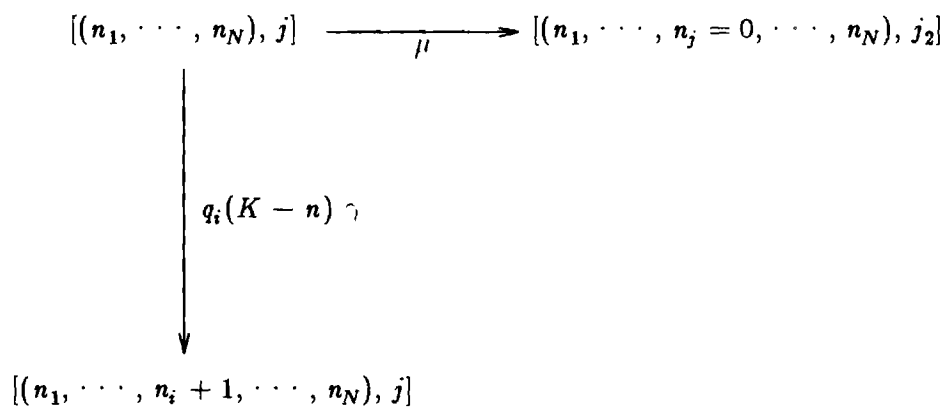
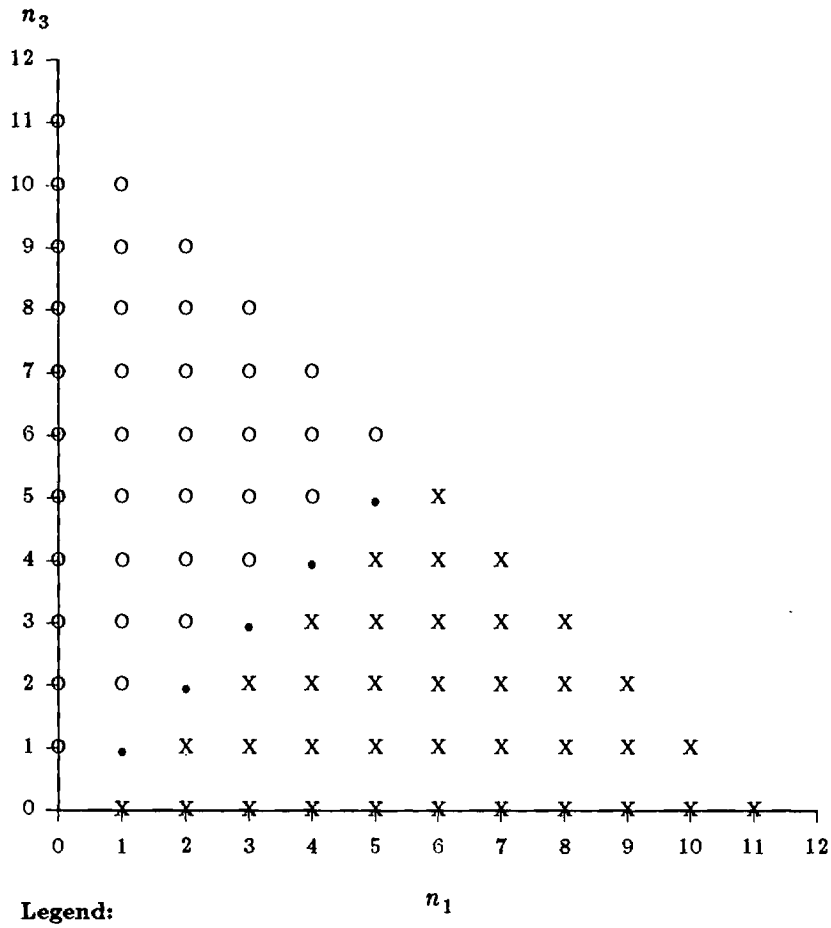


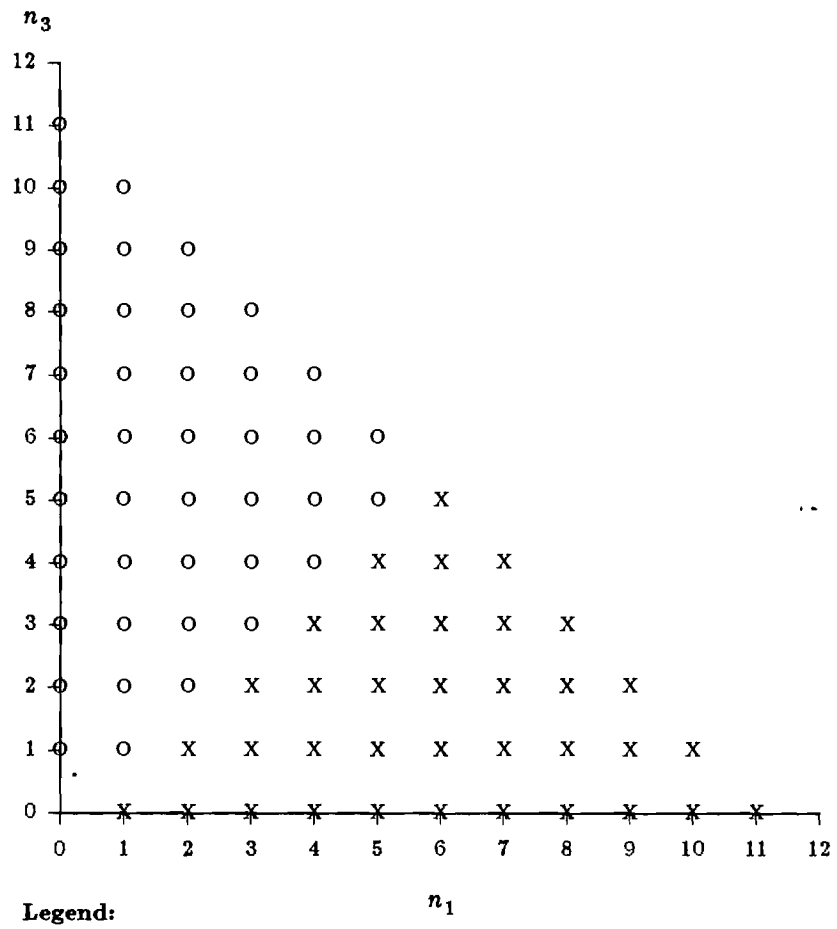
Figure 3. Markovian State Transitions.



Legend:

O	O	$d[(n_1,n_2,n_3),2] = 3$
•	•	$d[(n_1,n_2,n_3),2] = 1$ or 3
X	X	$d[(n_1,n_2,n_3),2] = 1$

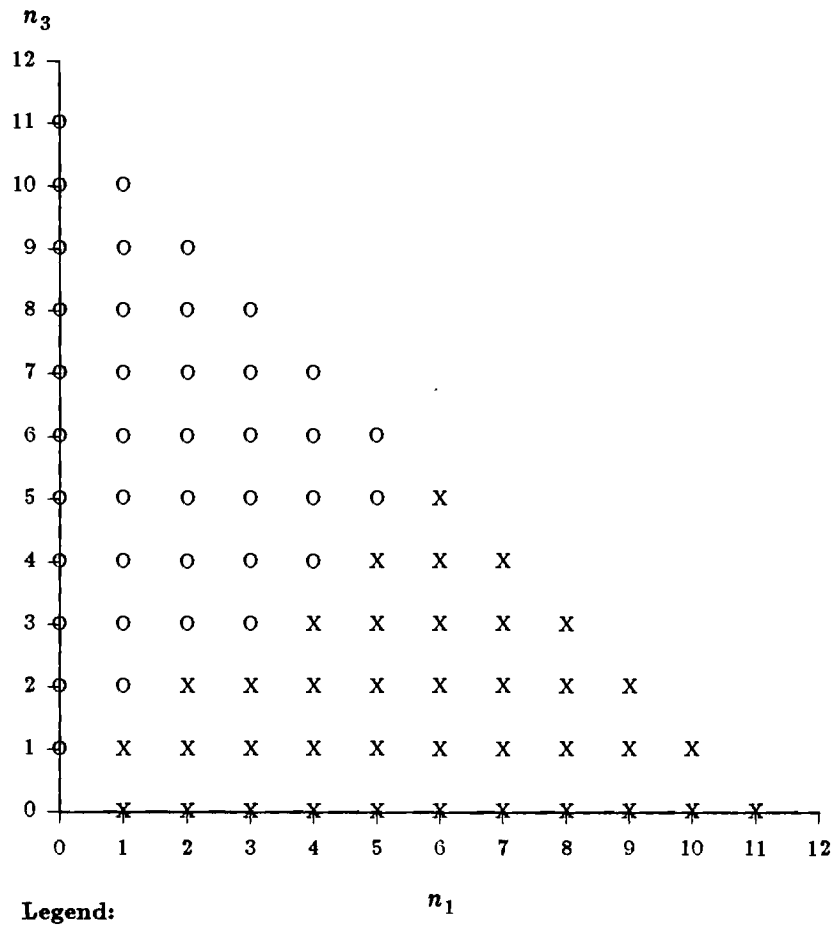
Figure 4. Optimal Scheduling Decisions for $q_1 = q_2 = q_3$.



Legend:

O	O	$d[(n_1,n_2,n_3),2] = 3$
X	X	$d[(n_1,n_2,n_3),2] = 1$

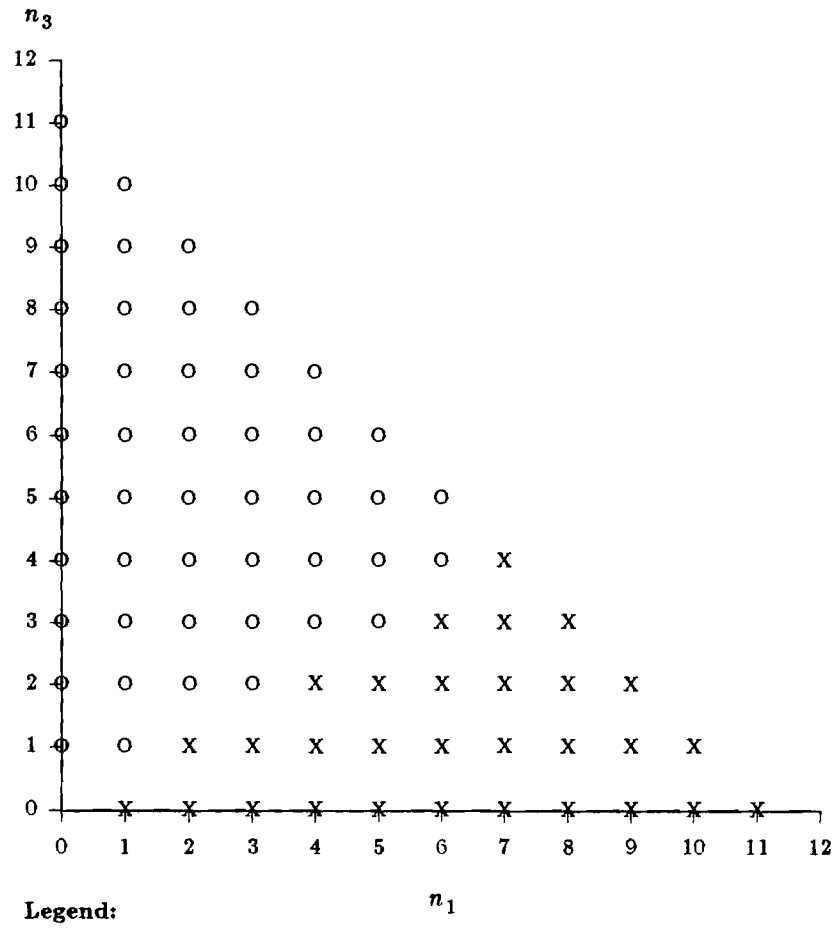
Figure 5. Optimal Scheduling Decisions for $P1$ and $P2$.
$(\frac{1}{\gamma} = 12; \frac{1}{\mu} = 1)$



Legend:

O	O	$d[(n_1,n_2,n_3),2] = 3$
X	X	$d[(n_1,n_2,n_3),2] = 1$

Figure 6. Optimal Scheduling Decisions for $P1$.
$(\frac{1}{\gamma} = 1.2; \frac{1}{\mu} = 1)$



Legend:

O	O	$d[(n_1,n_2,n_3),2] = 3$
X	X	$d[(n_1,n_2,n_3),2] = 1$

Figure 7. Optimal Scheduling Decisions for $P2$.
$(\frac{1}{\lambda} = 1.2; \frac{1}{\mu} = 1)$

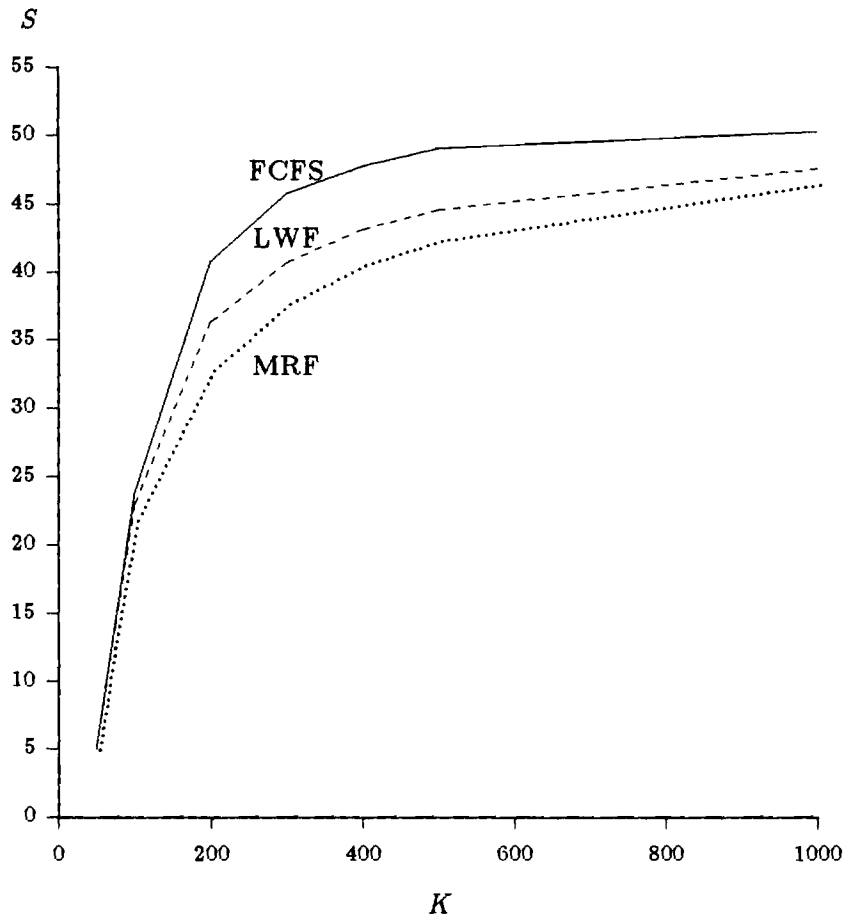


Figure 8. Mean Response Time versus Number of Users.

$(y = 0,\ N = 100,\ \mu = 1,\ \gamma^{-1} = 50)$

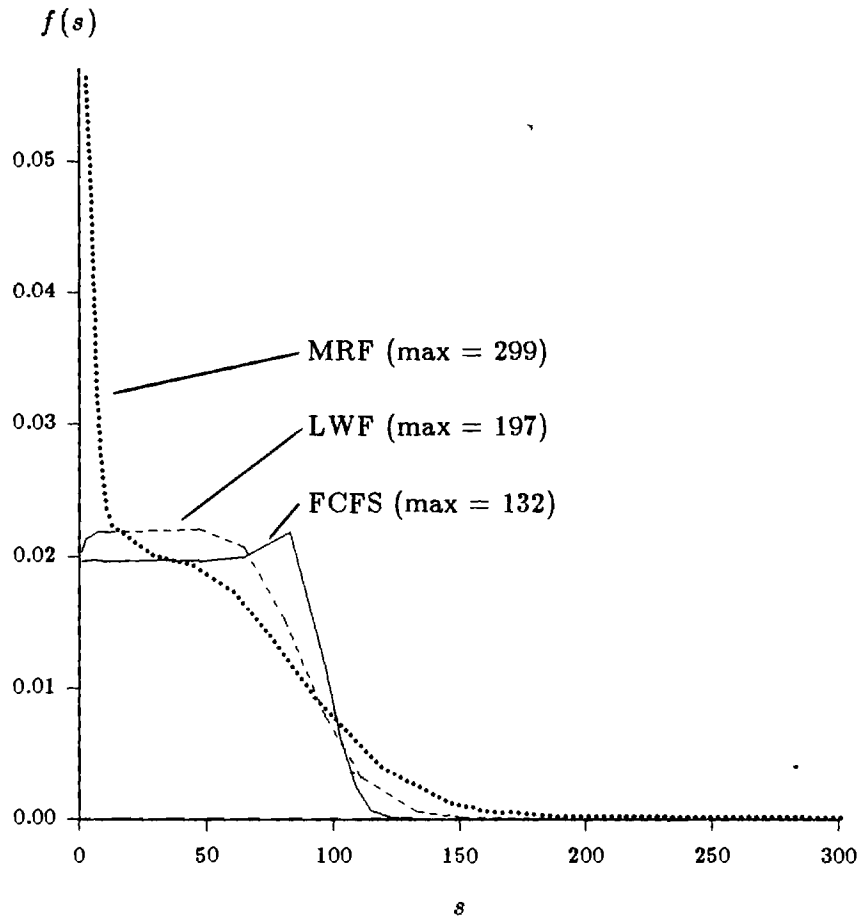


Figure 9. Probability Density Function (pdf) for Request Response Times (s).
($y = 0$, $K = 1000$, $N = 100$, $\mu = 1$, $\gamma^{-1} = 50$)

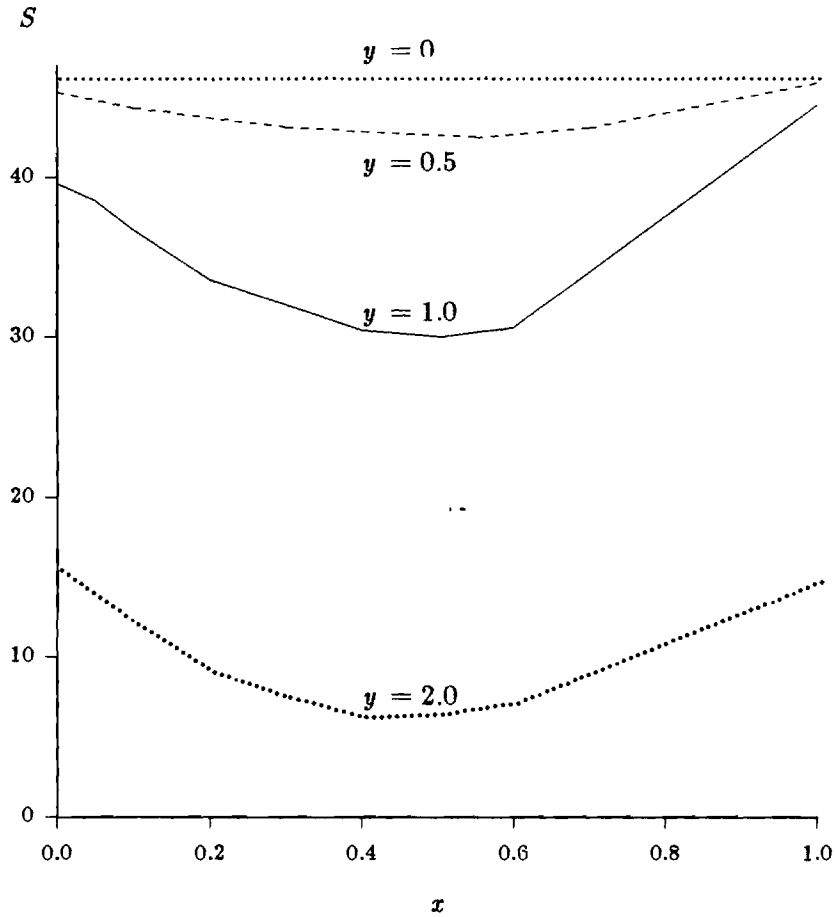


Figure 10. Mean Response Time versus MRF-B(x) Scheduling Parameter.
($K = 1000$, $N = 100$, $\mu = 1$, $\gamma^{-1} = 50$)

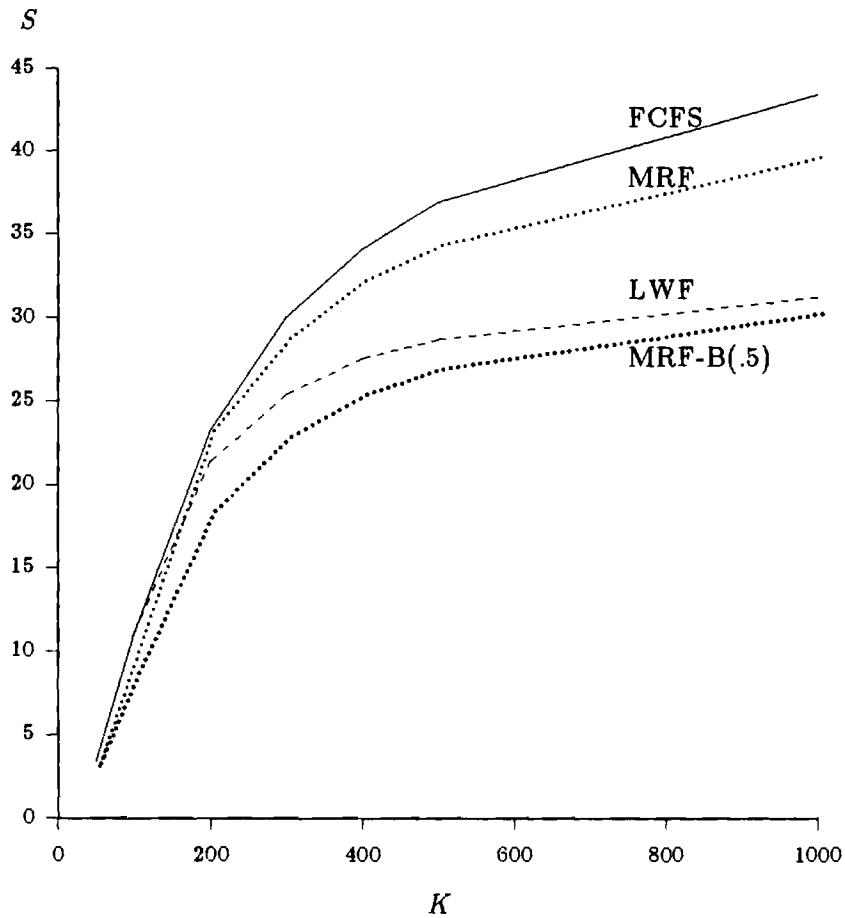


Figure 11. Mean Response Time versus Number of Users.

($y = 1$, $N = 100$, $\mu = 1$, $\gamma^{-1} = 50$)

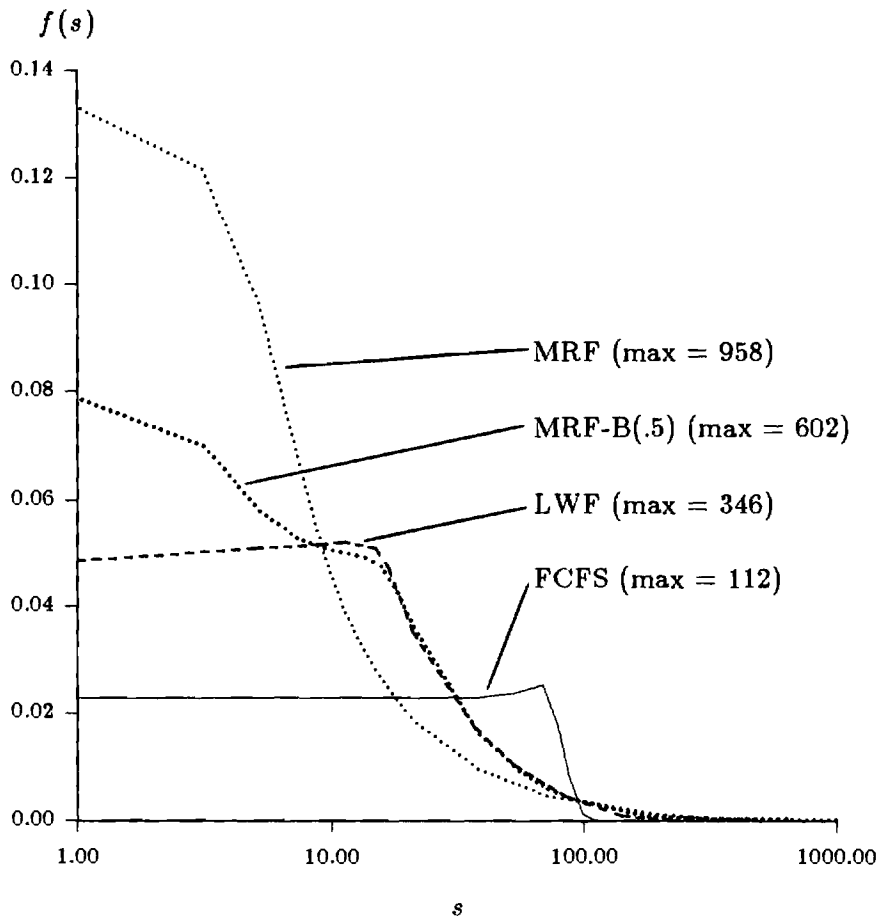


Figure 12. Probability Density Function (pdf) for Request Response Times (s).
($y = 1$, $K = 1000$, $N = 100$, $\mu = 1$, $\gamma^{-1} = 50$)

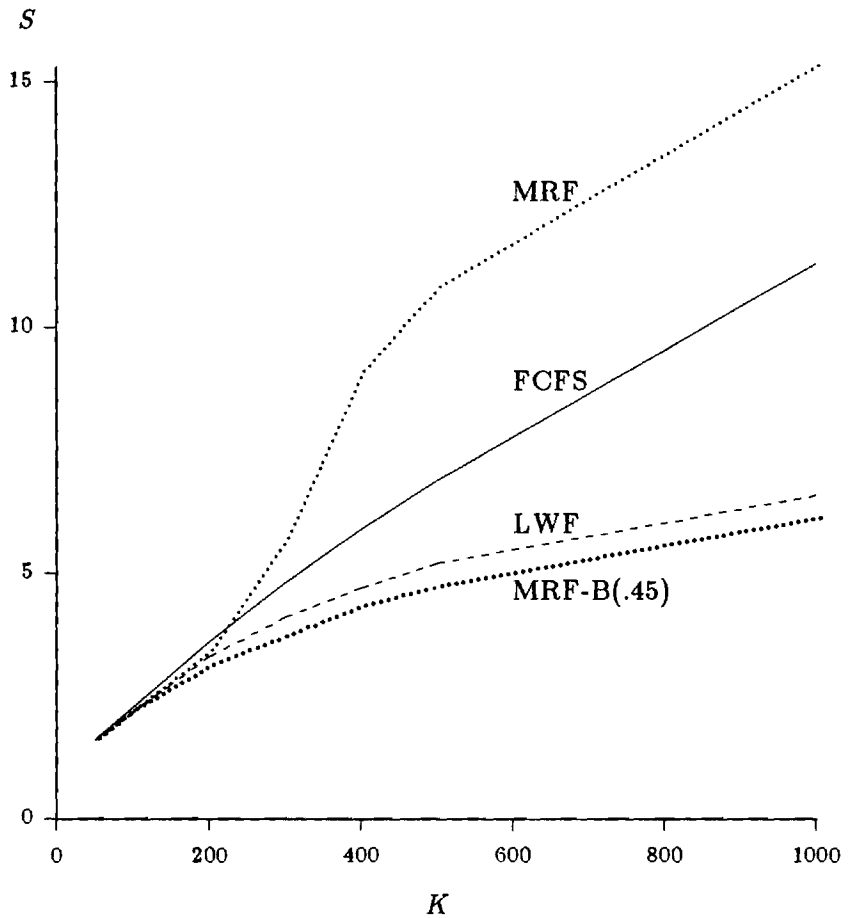


Figure 13. Mean Response Time versus Number of Users.
($y = 2$, $N = 100$, $\mu = 1$, $\gamma^{-1} = 50$)

	Update Data Structures	Determine Next Page to Process
FCFS	$O(1)$	$O(1)$
MRF,MRF-B(x)	$O(N)$	$O(1)$
LWF	$O(N)$	$O(1)$

Table 1. Time Complexities for Scheduling Functions

Using Multicast Communication to Locate Resources in a LAN-Based Distributed System*

Mustaque Ahamad
Mostafa H. Ammar
José M. Bernabéu Aubán
M. Yousef A. Khalidi

Technical Report GIT-ICS-87/44

Feb 18, 1988

Abstract

In this paper we present a resource location scheme using multicast communication. In the scheme, the universe of resource names is partitioned into a relatively small number of groups and each group is assigned a unique address. Nodes storing the locations of resources belonging to a particular group instruct their network interfaces to receive all location messages sent to the group address. To locate a resource, a node first determines the address of the group to which the the resource belongs (this can be accomplished via a well-known hash function), and a multicast message is then sent to the address. The algorithm performance is studied by means of simulation, and approximate closed form solutions are derived for systems operating at heavy and low loads. The scheme's performance is compared with that of broadcast, and it is shown that the proposed scheme performs much better than broadcast alone.

School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

*This work has been partially supported by NSF grants NCR-8604850, CCR-8619886, and CCR-8619886.

1 Introduction

The advantages offered by distributed systems include resource sharing, fault-tolerance and parallel execution of a computation. The programming of distributed systems is more complex than centralized ones due to the unavailability of the global state of the system. For example, in a dynamic system where resources can be migrated between nodes, a user must program an algorithm to find the current location of a resource needed by his or her computation. This can be avoided if users are provided with the abstraction of a unified system where the location of resources is transparent to them. Resources are referred to by names and, at runtime, the system determines the current location of a named resource.

Many schemes have been proposed for finding the location of a named resource. Conceptually, there exists a database that stores the associations between resource names and their locations. This database can be partitioned and stored at one or more nodes that are called *name servers* (some of the partitions may be stored at more than one name server). When a remote resource, R, needs to be accessed, the request for its location should be sent to a name server that stores R's location. The system must also implement algorithms to update the information stored by the name servers when resources are created or deleted or when they are migrated. To avoid this, the database can be distributed in such a way that a name server at a node maintains a list of only resources local to the node. In such a system a remote resource can be located by broadcasting its name, and having the node where the resource is located respond. This scheme is used in Clouds [DLS85] for locating remote objects. Broadcast can also be used when other schemes fail to locate a resource. One of the drawbacks to such an approach is that all nodes in the system have to participate in each location request.

In this paper, we explore the design of a distributed name server where multicast communication is used to locate the requested resource. Our goal is to design a location scheme that is simple from the point of view of a node that needs to find a resource but, at the same time, reduces the number of nodes that must participate in the location process. The availability of communications technology that supports multicast in the hardware provides further motivation for this work.

We associate a multicast address with each resource name and this address is used to communicate with the name server of the resource. Each node receives messages sent to multicast addresses corresponding to the resources whose locations are stored by the local name server. Since current network interfaces support a limited number of multicast addresses and the number of resources in the distributed system can be large, the resource

name to multicast address mapping is many-to-one. For such a system, we present the algorithms to be executed when a resource is created, deleted or a request is made for finding its location. We also study the performance of the multicast scheme and compare it with broadcast. The cost measure used is the number of nodes that process messages sent for finding a resource or for updating the information stored by name servers when resources are added or deleted. We use simulation and analytical techniques to determine the cost and demonstrate that the expected cost is much smaller for the multicast scheme when it is compared with broadcast.

We do not claim that to locate resources, a distributed system should use only the scheme proposed in this paper. It can be used in conjunction with other methods, e.g., hint tables to avoid broadcasting the request when a resource is not found at its expected location.

Section 2 describes related work and the system model is presented in section 3. The algorithms that implement the multicast scheme are described in section 4. We discuss the correctness of the algorithms in section 5 and consider the effect of node failures on the scheme in Section 6. Performance analysis and simulation results are described in section 7. We conclude the paper in section 8.

2 Related Work

Name servers are used for locating resources in many systems. In the Grapevine system [BLNS82], a resource name is of the type F.R where R is the name of a registry and F is the name of the resource in the R registry. Each registry has associated with it a collection of name servers. When the location of a name server for R is not known, it is found from a well-known registry wich is maintained in every name server. The Clearinghouse [OD83] system generalizes this by adding another level for naming.

If a resource is accessed at a node many times, its location can be cached so that the node does not have to consult with a remote name server each time the resource is used. Cached information is called *hints* and have been discussed in [Ter87,ABA88]. Since a resource can migrate, hints can be wrong and hence a name server should be located in that situation. In another scheme called *forwarding addresses* [Fow85], a node stores the address of the node where a resource residing at it has moved. A resource is located by following these addresses.

The broadcast scheme, where a message for finding a resource is sent to all nodes in

the network, is a special case of the scheme presented in [MV85] in which a node queries a subset of the nodes to find the location of a resource. In the V system [CM86], multicast is used to communicate with the name server nodes (this is done only when the resource is not found at the expected location and its name server is not known).

In the scheme presented in this paper, each node implements the scheme (it is possible to exclude certain nodes) and the set of nodes that receive a message sent to locate a particular resource depends on the resource name. Thus, the sets of nodes that process the location message for two different resources may be different (this will distribute the work and improve the performance). We study the relationship between the number of multicast groups supported by the hardware, the sizes of the multicast groups (the number of nodes that receive messages sent to a multicast address) and the number of resources in the system.

3 System Model

A distributed system is assumed to be a set of L (numbered from 1 to L) nodes and each node contains a set of resources which can be accessed by both local and remote nodes. Each resource has a unique name which is used by the users to refer to the resource. The set of resources residing at a node is dynamic: new resources can be created and existing ones can be deleted. Resources can also be migrated between nodes.

The nodes are connected by a broadcast network. A message sent over the network can be received by any subset of the nodes connected to it. A node consists of a processor (could also be a multiprocessor) with its own memory and a network interface that allows the processor to exchange messages with other nodes. The network interface receives messages transmitted over the network and performs address recognition to determine if an arriving message should be delivered to the processor. The interface is also responsible for transmitting messages sent by the processor.

We assume that the network interface can recognize the unique address associated with the node, the broadcast address and a set, Φ, of multicast addresses. A message sent to the multicast address m will be delivered to a processor only if m is in its Φ. A processor can change the membership of its own set Φ. However, the number of addresses in Φ cannot be more than M. Thus, at any point in time, a node can choose to receive multicast messages sent to at most M addresses. The restriction on the size of Φ holds for currently

available network interfaces. For example, the Digital UNIBUS Network Adapter, DEUNA[1] [Deu83], supports a maximum of 10 multicast addresses. We also assume that if a node sends a message which generates a response, the sender will receive the response in at most δ seconds. This allows the use of timeouts for deciding when not to wait for any more responses.

We assume that the operating system at each node, in addition to other functions, implements a *resource management subsystem*, RMS, and a *location subsystem*, LS. RMS handles the creation, deletion and migration of resources and stores information about all resources that are currently resident at its node. When a user needs to access a remote resource, RMS communicates with LS, which finds the current location of the remote resource. We assume that identical copies of RMS and LS execute at each node and RMS informs its local LS when a resource is created or deleted.

4 Location Subsystem

The LS executing at a node communicates with its local RMS and the LS at other nodes to implement a distributed name server. In this section, we describe the data structures maintained by each LS and its interface with the RMS. We also describe the algorithms executed when a resource is created, deleted or its location needs to be found. Since some of these functions can be executed concurrently at different nodes, their code must use some synchronization mechanism to assure atomicity when it is required. We do not include the code for synchronization to avoid the unnecessary complexity. Also, it is assumed that no node failures occur. The effect of failures on the scheme and their handling is described in a later section.

4.1 Location Subsystem Data Structures

Each LS maintains a directory of $< \textit{resource name}, \textit{node} >$ pairs and a multicast table. A multicast table entry consists of a multicast address and a count. A mapping (e.g., a hashing function), ω, which is well known, is used by LS to map a resource name to a multicast address. For each R such that $< R, i >$ is in the directory, there must be an entry in the multicast table with $\omega(R)$ as the address. The count field of this entry is the number of resources in the local directory that map to the address $\omega(R)$. Initially, the address field

[1] DEUNA and UNIBUS are trademarks of Digital Equipment Corporation.

in each entry of the table is set to a multicast address E which is not in the range of the mapping ω. As will be seen later, the resources whose names are stored in the directory can be both local as well as remote. Each address in the multicast table at a node is also added to its Φ and hence the size of the multicast table cannot exceed M. A multicast message sent by LS with m as its address will be received by a node if there is a resource, R, in its directory such that $\omega(R) = m$ (it is assumed that m belongs to a set of multicast addresses that are used only by the *location subsystem*).

4.2 Location Subsystem Calls

The RMS at a node not only calls the local LS for locating a remote resource, it also makes a call to LS when a resource is created or deleted. The functions implemented by LS that are called by RMS are defined below. To implement these functions, LS may have to communicate with its peers at other nodes. A message sent by LS contains its destination and source addresses, a type and data (if any) depending on the type of the message. The types of the messages used by LS and the data contained in them is described in the code for the functions. We assume that these functions are called at node i ($1 \leq i \leq L$).

- `AddResource`(R : ResourceName)

  This function is called by RMS when the resource R is created. This makes R available to remote nodes that can locate it by requesting their LS. Since the multicast table size is limited, the resource name and its location may have to be added to the directory at some other node. When it is not possible to add $< R, i >$ to the local directory, the `CreateSpace` function is called, and returns the node that can add $< R, i >$ to its directory and $\omega(R)$ to its multicast table. We describe the `CreateSpace` function later.

```
function AddResource (R : ResourceName)
    begin
        if ω(R) is in the multicast table then
            increment the count of the entry having address ω(R);
            add < R, i > to directory;
        else if there is an entry in the table with address E then
            change the address in the entry from E to ω(R) and make its count 1;
            add < R, i > to directory;
        else
```

j := CreateSpace(R);
if $i \neq j$ **then**
send a message of type *AddReq* to node j with $< R, i >$ as data;
else (* A free entry is created in local multicast table *)
change the address in the entry from E to $\omega(R)$ and make count 1;
add $< R, i >$ to directory;
end;

- FindResource(R : ResourceName)

This function is called by RMS to find the location of the remote resource R. If R exists at some node currently, the address of that node is returned by this function.

function FindResource (R : ResourceName)
begin
if $< R, j >$ in the directory **then**
return(j);
else
send a *FindReq* message to address $\omega(R)$ with R as data;
wait for *FindResp* message;
j := address of node sending the *FindResp* message;
return(j) ;
end;

- DeleteResource(R: ResourceName)

This function is called when RMS needs to delete R. RMS asks the local LS to find its location node, j, and delete the resource name and its multicast address at the node where they are stored. We do not consider the messages sent by RMS to its peer at node j to actually delete the resource.

function DeleteResource(R : ResourceName);
begin
if $< R, j >$ in directory **then**
delete $< R, j >$ from directory;
decrement the count in the entry with address $\omega(R)$ in the multicast table;

and when the count becomes 0, change $\omega(R)$ to E;
return(j)

else

send a *DeleteReq* message to $\omega(R)$ with R in the data field;
wait for a *DeleteResp* message returning node name j;
return(j)

end;

LS does not provide a function to be invoked when a resource is migrated. RMS can inform LS of the migration by deleting the resource at its current node and adding it at the new node by calling the functions described above.

4.3 Internal Functions of The Location Subsystem

We now describe the `CreateSpace` and the `MessageHandler` functions. These functions are internal because no other component of the system has access to them. Again, we assume that the functions are executed at node i.

- `CreateSpace`(R)

  The `CreateSpace` function is called by LS at node i when it cannot add $< R, i >$ to its directory because all addresses in the entries of the multicast table are different from $\omega(R)$ and E. Since *FindReq* messages for R are addressed to $\omega(R)$, the node where the location of R is stored must have $\omega(R)$ in its multicast table. The `CreateSpace` function finds a node where either $\omega(R)$ is in the multicast table or there is an entry with address equal to E. When this cannot be done, it creates an entry with address E at some node by moving resource names from the node's directory to some other node. The range of ω has to be restricted to assure that `CreateSpace` returns a node with this property. We discuss this in a later section.

  function `CreateSpace`(R)

  begin

  (* Check if some node has $\omega(R)$ in its multicast table *)
  send a *SpaceReq* message to $\omega(R)$;
  wait for *SpaceResp* message to arrive for δ time;
  if one or more *SpaceResp* messages arrive **then**

choose one and let j be the sender of the chosen message;
return(j);
(* Check if some node has address E in its multicast table *)
send a *SpaceReq* message to E;
wait for *SpaceResp* messages to arrive for δ time;
if one or more *SpaceResp* messages arrive **then**
choose one and let j be the sender of the chosen message;
return(j);
(* No node has $\omega(R)$ in its table and all tables are full *)
send a *TableReq* message to the broadcast address;
wait for *TableResp* messages to arrive for δ time;
let j and k be two nodes such that multicast tables received from
them in the *TableResp* messages have a common multicast address[2] m.;
send a *MoveDirEntryReq* message to j with k and m in the data field;
wait for a *MoveDirEntryResp* message;
return(j) ;

- `MessageHandler`(msg)

The `MessageHandler` function is executed by LS at node i when a request message arrives for LS. This message may have been sent to the unicast address of i or a multicast or the broadcast address. Since only request messages arrive asynchronously, we show the handling of these messages. The response messages are received when LS sends a request message and their handling is described in the code of the functions.

function `MessageHandler`(msg : Message)
begin
j := sender of msg;
case msg.type **of**
AddReq: $< R, k >$:= data received in msg;
if $\omega(R)$ is the address in an entry in the table **then**
increment the count in the entry with address $\omega(R)$;
add $< R, k >$ to directory;
else if there in an entry with address E **then**
change the address in the entry from E to $\omega(R)$;

[2] An alternative way of getting two nodes which share a multicast address is to poll nodes one at a time until two nodes with a common address are identified.

make its count 1;
add $< R, k >$ to the directory;

FindReq: R := resource name received in msg;
if $< R, k >$ in the directory **then**
send k in a *FindResp* message to j;

DeleteReq: R := resource name received in msg;
if $< R, k >$ in directory **then**
delete $< R, k >$ from directory;
decrement the count in the multicast table entry
having address $\omega(R)$;
if count becomes 0 **then** change $\omega(R)$ to E;
send k in a *DeleteResp* message to j;

SpaceReq: send a *SpaceResp* message to j;

TableReq: send the multicast table in a *TableResp* message to j;

MoveDirEntryReq:
m := multicast address received in msg;
k := node address received in msg;
for each $< R, l >$ in the directory such that $\omega(R) = m$ **do**
send an *AddReq* message to k with $< R, l >$ as data;
change $\omega(R)$ to E in the multicast table;
send *MoveDirEntryResp* message to j;

5 Correctness

The correctness requirement for the multicast based scheme is that when a resource exists (it has been added by calling `AddResource`(R) and it has not been deleted by calling `DeleteResource`(R)), then executing `FindResource`(R) at any node must return the current location of R. Let i be the node where `FindResource` is executed and let j be the current location of R. If $< R, j >$ is not in the directory at node i then a *FindReq* message is sent to $\omega(R)$. Thus, the location of R will be returned by the `FindResource` call if the node where $< R, j >$ is stored in the directory has $\omega(R)$ in its multicast table. This will guarantee that the *FindReq* message for R is received by the node that stores its location. Since $< R, j >$ is added to the directory at a node only when either $\omega(R)$ is in the multicast table or there is an entry with address E which is changed to $\omega(R)$ (`AddResource` function and handling of *AddReq* message in `MessageHandler`), the correctness follows if we can

demonstrate that $< R, j >$ is added to the directory at some node as a result of executing `AddResource`.

If $\omega(R)$ or E is the address in an entry of the multicast table at node i when `AddResource` is executed, $< R, i >$ is added to the local directory. Otherwise, $< R, i >$ is sent to node j which is returned by the `CreateSpace` function. If j is the address of the chosen node that responded to the *SpaceReq* message sent to either $\omega(R)$ or E then $< R, i >$ is added to the directory at the responding node. When no nodes respond to the *SpaceReq* messages sent to these addresses, then multicast tables at all nodes are full (there is no entry with E as the address) and none of the tables has an entry with the address $\omega(R)$. In this case, all multicast tables are collected at node i and two tables having a common multicast address are found. To guarantee that there exist two such tables, we need to restrict, K, the range of ω. Since the multicast table size is M and there are L nodes, if $K \leq L \cdot M$ then two tables will have a common address when all tables are full and none of them has the address $\omega(R)$. This follows because otherwise $K \geq L \cdot M + 1$ (all addresses in the multicast tables are distinct and different from $\omega(R)$) which is a contradiction.

Once two nodes such that their multicast tables have a common address, a, are found, the entry containing a at one node is freed by sending all resource names mapping to the address a to the other node that sent the table with address a in it. The resource name entries deleted from the directory of one node are added at the other node because the multicast address corresponding to the resource names is in the table at the other node. R is added to the directory at the node where the free entry is created. Thus, when `AddResource`(R) is called, the name and location of R are added to the directory of some node which has $\omega(R)$ in its multicast table.

6 Accommodating Node Failures

In the scheme presented, when RMS creates a resource R at node i and calls `AddResource`(R), the location of R may be stored at a node other than i. This happens when the multicast address table at node i is full and does not have $\omega(R)$ in any of its entries. When the location of R is stored at node j and $i \neq j$, the failure of j can make the resource unavailable to remote nodes even when i (the node where R exists) is operational. This is due to the fact that LS will not be able to find the location of R because the node where R's location is stored has failed. Thus, no *FindResp* message will be received when `FindResource`(R) is executed. Node failures also affect the `AddResource` function when no two multicast tables at the currently operational nodes have a common address. In this case, the `CreateSpace` function

may fail because the location of R cannot be stored in the directory at any node. When $K \leq L \cdot M - f \cdot M$ then only the failure of more than f nodes will cause the `CreateSpace` function to fail due to the reason that no two multicast tables at the currently operational nodes have a common multicast address.

If we require that LS find a resource if it is at an operational node then we have to extend the scheme. Broadcast can be used by the scheme when failures occur. Thus, when a *FindResp* message is not received in δ time, the *FindReq* message is broadcast. Also, we require that the directory maintained by LS at each node store the $< R, i >$ pairs for all local resources even when $\omega(R)$ is not in the local multicast table (a remote node table has $\omega(R)$ and the directory at that node stores R's location), then the node where R exists will respond to the broadcast *FindReq* message. Thus, when no response is received when *FindReq* is sent to $\omega(R)$, the node storing R's location has failed and broadcast is used to locate R (notice that a non-existent resource will also cause a broadcast request to be sent but we do not consider the behavior of the scheme when it is requested to locate such resources.)

`AddResource` can also be made failure-resilient in a trivial way. Since the directory at the node where R is created stores its location (to locate R in case of failures), `AddResource`(R) can return even when $\omega(R)$ cannot be added to the multicast table at any node. When a remote LS wants to find the location of R, the *FindReq* message sent to $\omega(R)$ will fail and then the request will be broadcast. In that case, the node where R exists will respond with its location. We do not consider the algorithm to be used when recovery of failed nodes will allow such resources to be added to the directory of a node having $\omega(R)$ in its table.

7 Performance Study

To study the performance of the location scheme presented above we will use a simulation model of a system that uses the multicast location algorithm. The simulation results will provide us with an understanding of how the performance of the proposed scheme is affected by various parameters and how it compares with the use of broadcast to locate a resource. We will also present analytic results for light and heavy load approximations.

7.1 Simulation Model

For the purpose of the simulation we make the assumption that *find*, *delete* and *add* operations occur independently of each other. Requests to *find* and *delete* a particular resource are only allowed when the resource has been added but not yet deleted. Resources are added to the system as a Poisson process with rate γ. An *add* request is equally likely to arrive at any of the nodes. A resource being added is equally likely to have its name mapped to any address (this is a property of the function ω). Thus the total arrival rate of *add* requests per node per address is $\frac{\gamma}{LK}$ (recall that L = number of nodes, K = number of multicast addresses in the range of ω, and M = size of multicast tables). Once a resource is added, it will reside in a node for a time that is exponentially distributed with rate μ. A *delete* request for a particular resource is equally likely to occur at any node in the system. Once a resource has been added, *find* requests are generated for it at a rate λ until it is deleted. The interarrival time of find requests for a particular resource is exponentially distributed and a *find* request is equally likely to arrive at any node in the system.

We are interested in studying the system in the steady state, and in that state, the rate of resources leaving the system will be γ, and the average time spent by a resource in the system is given by $\frac{1}{\mu}$. Thus, the average number of resources in the system, $\bar{r}$, can easily be computed by applying Little's Law[Lit61].

$$\bar{r} = \frac{\gamma}{\mu} \tag{1}$$

The simulation closely follows the steps of the algorithms presented in section 4. In the definition of the `CreateSpace` function, three phases can be distinguished. In the first phase, a *SpaceReq* message is sent to a multicast address (different from E) and one of the nodes responding to it is selected. In the simulation, it is equally likely that any particular node be selected from the set of nodes having the multicast address in their tables. In the second phase, one of the nodes with empty multicast table entries has to be selected. Again, any node is equally likely to be selected. Finally in the third phase both a multicast address and two nodes belonging to its multicast group have to be selected. It is equally likely that any particular multicast address will be selected out of those which are in tables at more than one node. Any pair of nodes with that address is also equally likely to be chosen.

We assume in the discussion that the system will be fault-free. In particular it will always be possible to add a new resource, and resources for which *find*'s and/or *delete*'s arrive, must have already been added.

7.1.1 Cost Calculation

We describe the performance of the multicast scheme in terms of the cost of certain operations. This cost is defined as the number of messages processed by nodes in the system. Thus, for example, if during an operation a message is sent to a multicast group consisting of 3 nodes, and in turn one of the nodes sends back an acknowledgement message, the cost of this operation would be 4 under the proposed metric. We think this metric properly reflects the amount of CPU usage in the system. In calculating our costs we are concerned only with the function that terminates by returning the location of the resource to the RMS. Additions and deletions of resource references are considered, however the costs of addition, deletion or usage of the resource itself are not.

We will now formally present how the costs are computed for each one of the *location subsystem* operations. In what follows it is assumed that the operation is requested for resource R, and originates at node i. a will represent the multicast address to which R maps ($\omega(R)$) and $\mathcal{N}(a)$ will represent the group of nodes having a in their multicast tables ($N(a)$ will denote the number of nodes in $\mathcal{N}(a)$). $\mathcal{N}(E)$ will be used to represent the set of nodes containing at least one free entry in their multicast table ($N(E)$ will represent its cardinality).

find:

- If R is local to i, the cost is zero.
- If the reference (location information) for R is maintained at i, the cost is also zero.
- If none of the above is true and $i \notin \mathcal{N}(a)$, then the cost will be $N(a)+1$, that is, each node in the multicast group will process a message, and node i will process the acknowledgement.
- If none of the above is true but $i \in \mathcal{N}(a)$, then the cost will be $N(a)$, that is, all nodes in the multicast group, except i, will process one message, and node i will process the acknowledgement.

delete:

- If the reference for R is maintained at i, the cost will be zero.
- If the above is not true and $i \notin \mathcal{N}(a)$ then the cost is $N(a)+1$.
- If the above is not true and $i \in \mathcal{N}(a)$ then the cost is $N(a)$.

add:

- If $i \in \mathcal{N}(a)$ the cost is zero.
- If i has a free entry in its multicast table, the cost is also zero.
- If the above is not true and $\mathcal{N}(a)$ is not empty, the cost is $2 \cdot N(a) + 1$.
- If the above is not true and there is some node in the system with an empty entry in its multicast table, then the cost is $2 \cdot N(E) + 1$.
- If the above is not true, the cost will be $2 \cdot (L-1) + 3$, explained by the fact that it is necessary to perform a broadcast to $(L-1)$ nodes, and receive all responses. Node i then sends messages to the two selected nodes, after which one of the nodes passes its table to the other with one message.

From the above description it is clear that the costs of the *find* and *delete* operations will vary similarly, being indeed very close in value. The average cost of *find* operations will actually be slightly smaller than that of *delete* operations because a *find* operation will have zero value in two cases: when the resource is local or when its reference is kept locally.

7.2 Cost of Broadcast

With the same assumptions about system behavior as we made above, we can find the average cost for each operation type (and the total combined average for all types) when broadcast is used as the only location method.

We first note that a *find* or *delete* operation will have to proceed as follows:

- Search for the resource locally, if found finish the search with cost zero.
- If resource is not found locally, broadcast its name to the rest of the nodes and wait for the node having the resource to answer. Finish with cost L.

The probability that a resource, chosen at random, be local will be given by $\frac{1}{L}$, thus the average cost of a *find* (or *delete*) operation will be given by,

$$C_f = C_d = L \cdot \frac{L-1}{L} = L - 1 \tag{2}$$

The cost of additions will be always zero, because the resource reference is stored only locally. To find the total combined cost for the three types of operations, we first have to

discuss the probability with which each type of operation is requested. In the steady state, the rate of *add* and *delete* operations will be the same, and equal to γ. On the other hand, the rate of *find* operations will be given by $\bar{r}\lambda$, where $\bar{r}$ is as given by equation (1). Thus the total rate of arrival of operations is $2 \cdot \gamma + \bar{r}\lambda$. The probability that an operation is of a certain type will be given by the ratio of the operation's rate to the total rate. Thus,

$$\begin{aligned} P[find] &= \frac{\bar{r}\lambda}{2 \cdot \gamma + \bar{r}\lambda} = \frac{\lambda}{\lambda + 2 \cdot \mu} \\ P[delete] &= \frac{\gamma}{2 \cdot \gamma + \bar{r}\lambda} = \frac{\mu}{\lambda + 2 \cdot \mu} \\ P[add] &= \frac{\gamma}{2 \cdot \gamma + \bar{r}\lambda} = \frac{\mu}{\lambda + 2 \cdot \mu} \end{aligned} \tag{3}$$

(Note that the expressions above are independent of the location scheme used.)

Thus the average combined cost for broadcast will be,

$$C_B = (L - 1)\frac{\lambda + \mu}{\lambda + 2 \cdot \mu} \tag{4}$$

7.3 Simulation Results

We performed two sets of simulations. In the first set, the simulations were run for a system consisting of 20 nodes in which each node's multicast table could hold up to 10 multicast addresses. In the second set, the value of K was fixed to 100, and the value of M varied from 5 (its minimum) to 100.

For the first set, different values for K have been considered, covering all its possible range (notice that it is necessary that $K \leq L \cdot M$). The maximum load (average number of resources in the system) considered was 2000 (that is 100 resources per node on the average when the multicast table size, M, is 10). Due to the fact that the cost of *delete* operations varies similarly to the cost of *find*'s, we have only shown the latter's costs.

In figure 1, we show the variation of the average cost of *find* operations. We plot the variations for several values of K. It can be observed that the costs reach a definite asymptotic value at high loads, and this value is reached relatively fast as the load is increased. It can also be seen that for large K (close to the maximum), there is a relative maximum in the cost curve (although it is not very pronounced). To understand this behavior, we have to consider what happens when the load varies from 1 to 2000. We start by pointing out that with our cost measure, the average cost of a *find* operation will increase with the number of nodes receiving messages sent to a given multicast address. The larger the number of multicast addresses, the smaller the number of nodes with a

given address. At load 1 there is, on the average, a number of nodes close to one which contain a given multicast address. As the number of resources in the system increases, the number of nodes containing resources that map to a given multicast address will increase while there is enough room in the tables to store the multicast addresses of all existing resources. Thus the cost of *find* requests will also increase. As the multicast tables start getting full and $K > M$, the multicast addresses will compete with each other for a place in the tables as a result of calls to the `CreateSpace` function. This will in general decrease the number of nodes in a given multicast group: references of resources that map to a particular address will be moved by using the *MoveDirectory* message and will be collected at a small number of nodes, thus decreasing the cost of a *find*. When $K \leq M$ the cost curves are monotonically increasing. This is because there is never competition between the multicast addresses, and, in the limit, all addresses are in the multicast tables of all the nodes, thus making any multicast message equal in cost to a broadcast.

As shown in figure 1, the larger the number of addresses, K, the lower the cost of *find*. This is a direct consequence of the fact that increasing the number of multicast addresses reduces the number of multicast table entries available per address, thus reducing the number of nodes in a particular multicast group. For the system being considered, the average cost of a *find* operation when only broadcast is used to locate objects is given by equation (2) and equal to 19. It can be seen that even for relatively low values of K ($K = 20$), the cost of using the multicast scheme is slightly more than half that of broadcast for heavy loads (it is even lower for low loads). When K is incremented to 50, the cost reduces to approximately one fifth of the broadcast cost. Thus the multicast method compares very favorably with respect to broadcast for *find* operations (the same can be said about *delete* operations).

Figure 2 plots the average cost of *add* operations versus the average number of resources in the system. For $K \leq M$ this cost will be zero (there is always room in the multicast table to store the address of a new resource). For any given K, the cost seems to vary similarly to the cost of *find*'s. The biggest difference consists of the fact that at low loads, the larger K, the larger the cost, whereas at high loads the opposite is true. This happens because at loads high enough so that addresses have already started to compete for multicast table entries, but low enough that the number of nodes in each multicast group has not yet been balanced, the likelihood of a totally new address coming in the system is high, thus forcing the execution of the `CreateSpace` function up to its second phase. Once the number of nodes per multicast address starts balancing, however, all multicast addresses will have at least one entry in the multicast tables, and the larger the K, the lesser the number

of nodes containing any particular address. Thus, the cost of executing the `CreateSpace` protocol decreases with K, and, according to the figures, although the probability that the `CreateSpace` protocol be executed increases with K, its cost becomes low enough as to make it cheaper for higher values of K. For *add* operations, the multicast scheme clearly performs worse than broadcast, whose cost is zero.

We call the ratio of *add* request rate to *find* request rate the *operation mix*. The actual operation mix will not affect the costs of *find*, *delete* and *add* operations at any given load, however it will affect the overall average cost for all operation types. In figure 3 we show the variation of the overall average cost for all operation types for a system in which the operation mix is $1 : 40$. It can be seen that the variation of the costs follows closely the one observed in figure 1, which is due to the fact that *find* operations are the ones contributing most to the overall cost. The average overall cost for broadcast as given by (4) would be slightly less than 19, and the overall cost of the multicast scheme still is only slightly higher that half the overall cost of the broadcast scheme for $K = 20$, and much lower for higher values of K.

In figure 4 we also show the variation of the total cost but for a different operation mix $1 : 1$. In this figure, the larger influence of the cost of *add* operations can be observed in the way the cost curve for $K = 200$ is disturbed. For this mix, the overall average cost of the broadcast scheme improves, becoming slightly more than 12 (12.67). For $K = 20$, the multicast scheme still has a total cost below 10, and for $K = 50$, the cost becomes slightly more than 5.

In figure 5 we plot the variation of the average cost of *find* operations against the number of multicast addresses, K, for some values of the load. It can be seen that the cost falls sharply as K increases. It can also be observed that for K close to its maximum (200) higher loads lead to somewhat lower costs of *find*.

In figure 6 we plot the variation of the cost of *add* operations against the number of multicast addresses, K. In this figure, the effects seen in figure 2 are made more apparent: at medium loads, the larger K the larger the cost of *add* operations.

In figure 7 the variation of the average cost of *find* operations is plotted as M, the size of a node's multicast table, increases. The value of K for all curves is set to 100. At low loads, the cost does not seem to depend on the value of M (this is in agreement with the results obtained for the low load approximation, see next section). In general, for all loads, the cost will increase until a certain value for M is reached. Thus, by increasing M sufficiently, the system can be made to work in the "low load range".

A similar effect can be observed in figure 8, where the cost of *add* operations is plotted against M. Here, again, we observe that for sufficiently large M the system starts operating in the "low load range" (characterized by the cost of *add* being close to zero). This value for M coincides with the one observed on the plot for the cost of *find*.

7.4 Approximate Analysis

The results of the simulation indicate that the system seems to be operating mainly in two modes: at low loads, the cost increases rapidly, whereas after a certain value of the load its behavior changes radically and the system stabilizes with an almost constant cost. This suggests a description of the system's behavior at heavy and low loads will be useful to understand the system's overall behavior.

It is possible to provide models which will approximate the behavior of the algorithms for the low loads. Such analysis will provide us with closed form expressions for the costs. It is also possible to obtain models which provide upper and lower bounds on the costs when the system is operating at heavy loads.

In Appendix A we derive the expressions for the approximate costs at low loads. This is achieved by assuming that there is always room in the multicast tables to store the address to which a resource maps and, thus, all resource references are stored at the node where the resource resides. We obtain the following results for the average costs (note: C_a, C_d and C_f stand for the average costs of *add*, *delete* and *find* operations, respectively. C represents the overall average cost).

$$\begin{aligned} C_a &= 0 \\ C_f &= (2 + (L-2)(1 - e^{-\frac{r}{KL}}))\frac{L-1}{L} \\ C_d &= C_f \\ C &= \frac{\mu + \lambda}{\lambda + 2 \cdot \mu} C_f \end{aligned} \tag{5}$$

Notice that the costs do not depend on the value of M. Also note that as $\bar{r} \rightarrow \infty$, the costs in (5) approach those derived in equations (2) and (4).

In Appendix A we also derive upper and lower bounds for the different costs in the heavy load limit. Under this limit the system is assumed to have reached a given configuration for its multicast tables. In this configuration, all multicast addresses are stored in at least one entry of some multicast table. The configuration, once reached, does not change unless

the load decreases. For *add* and *find* operations we derived upper and lower bounds for the limit of the cost at heavy loads. Denoting the upper and lower bounds of an operation o by C_o^u and C_o^l respectively, we have,

$$\begin{aligned}
C_a^u &= (2L-1)\frac{M}{K} + 1 - \frac{2}{LK}\left[LM + \lfloor\frac{LM}{K}\rfloor\left(2LM - K(\lfloor\frac{LM}{K}\rfloor + 1)\right)\right] \\
C_a^l &= (2L-1)\frac{M}{K} + 1 - \frac{2}{LK}(qL^2 + (p+1)^2 + K - q - 1) \\
C_f^u &= (L-1)\frac{K + M(L-2)}{LK} - \frac{M}{LK} + \frac{1}{L^2K}(qL^2 + (p+1)^2 + K - q - 1) \\
C_f^l &= (L-1)\frac{K + M(L-2)}{LK} - \frac{M}{LK} \\
&+ \frac{1}{L^2K}\left[LM + \lfloor\frac{LM}{K}\rfloor\left(2LM - K(\lfloor\frac{LM}{K}\rfloor + 1)\right)\right]
\end{aligned}$$

Where $q = \lfloor\frac{LM-K}{L-1}\rfloor$ and $p = LM - K - q(L-1)$. For *delete* operations we were able to obtain the heavy load limit given by,

$$C_d = (L-2)\frac{M}{K} + 1$$

In figure 9 we plot the low load limit value and the heavy load bounds for the cost of find for $K = 20$. We can see that the low load limit fits the simulation curve at low loads. As the load increases, the simulation curve eventually enters the zone between the upper and lower bound approximations. A close match with the simulation results has also been observed for the cost of the other operations and for all the values of K we have studied.

8 Concluding Remarks

In this paper we presented the algorithms necessary to implement a simple location scheme based on multicast communication. To analyze its performance, a simulation model was developed which closely followed the steps of the algorithms. The simulation results showed that the scheme had a lower cost than broadcast alone. In order to predict the costs of the scheme for cases not included in the simulation, analytic results were obtained which approximate the behavior of the system at low loads, and provide tight upper and lower bounds on the costs incurred when using this location scheme on systems operating with a large number of resources.

In all cases considered, the cost of *find* operations using the multicast scheme is lower than if broadcast were used instead. Even when the number of multicast addresses is less than or equal to the number of entries in the multicast table, the multicast scheme presented

in this paper has a lower cost than broadcast for low values of the load. Even though the cost of *add* will always be worse for the multicast scheme (for broadcast its cost will always be zero), the overall cost still favors the multicast scheme for large enough values of K.

In general it can be concluded that the scheme presented is most advantageous for large values of K. However the range for K is constrained by the value of M, the number of entries in the multicast table. The larger the size of the multicast tables, the larger K can be made, which will make the scheme presented in this paper cheaper. The above discussion seems to indicate that given a system with a certain limit M on the size of its multicast tables, K should be set to $L \cdot M$ in order to achieve the best performance. However, as discussed in section 6, doing so will increase the vulnerability of the scheme to failures.

Appendix A: Approximate Analysis

The approximations in this appendix are based on the assumptions used for our simulation model as described in section 7.1. Any further assumptions are explained as needed.

A.1 Low Load Approximation

In this approximation we assume that there is always room to insert a new entry for the address of a newly created resource in the multicast table of a node. This will be a good approximation when the number of resources is small. It will be an exact model when $K \leq M$. Under this approximation, the way resources are added at each node does not depend on the way they are added at any other node.

When it is always possible to store the reference to a resource at the same node the resource is, we can model each node as a $M/M/\infty$ queue [Kle75] with K classes. The total arrival rate, γ, will be equally divided between all nodes, and for each node it will also be equally divided among all multicast groups, thus giving an arrival rate per class into each one of the queues of $\frac{\gamma}{LK}$. Under this model, the cost of *add* will, of course, be zero, because one of the assumptions is that there is always space in the multicast table to store the multicast address of any new resource, and this situation has a zero cost for *add*. The cost of a *find* operation for a resource which maps to multicast address a will depend only on the number of queues with a non–empty population of customers of class a at the moment the *find* occurred. The same can be said about *delete* operations.

Letting $\underline{n} = (n_1, \ldots, n_L)$, where n_i is the number of resources mapping to multicast address a which are at node i, $\underline{n}$ will contain all information we need to know about the state. In the following discussion, when we talk about the state of the system we will be assuming that it is described by a vector $\underline{n}$. We will also use the name "resource" to indicate a resource mapping to multicast address a (unless otherwise mentioned). Thus the steady state probabilities will be given by [BCMP75],

$$P(\underline{n}) = \prod_{i=1}^{L} P_i(n_i) = e^{-\bar{n}} \left(\frac{\bar{n}}{L}\right)^n \prod_{i=1}^{L} \frac{1}{n_i!} \tag{A1}$$

where $P_i(n_i) = e^{-\frac{\gamma}{LK\mu}} (\frac{\gamma}{LK\mu})^{n_i} \frac{1}{n_i!}$ is the marginal probability that node i contains n_i resources; $\bar{n} = \frac{\gamma}{K\mu}$ is the average number of resources in the system (for all nodes), and $n = \sum_{i=1}^{L} n_i$.

In order to determine the average cost of a *delete* operation it is necessary to determine

the probability that a delete finds the system in a given state. Let $D_i(\underline{n},t)$ be the probability that a delete finds the system in state $\underline{n}$ at time t and transforms it to state $\underline{n}-\underline{1}_i$, where $\underline{n}-\underline{1}_i$ stands for the state obtained from $\underline{n}$ by subtracting one unit from n_i. Also, let $A(t,t+\Delta t)$ stand for the event "A delete occurs between times t and $t+\Delta t$", then,

$$\begin{aligned} D_i(\underline{n},t) &= \lim_{\Delta t\to 0} P[s(t+\Delta t)=\underline{n}-\underline{1}_i; s(t)=\underline{n}|A(t,t+\Delta t)] \\ &= \lim_{\Delta t\to 0} \frac{P[s(t+\Delta t)=\underline{n}-\underline{1}_i; s(t)=\underline{n}, A(t,t+\Delta t)]}{P[A(t,t+\Delta t)]} \\ &= \lim_{\Delta(t)\to 0} \frac{P[s(t+\Delta t)=\underline{n}-\underline{1}_i; A(t,t+\Delta t)|s(t)=\underline{n}]\cdot P[s(t)=\underline{n}]}{P[A(t,t+\Delta t)]} \end{aligned}$$

Now, considering that

$$\begin{aligned} P[A(t,t+\Delta t)] &= \bar{n}\mu\Delta t + o(\Delta t^2) \\ P[s(t+\Delta t)=\underline{n}-\underline{1}_i, A(t,t+\Delta t)|s(t)=\underline{n}] &= n_i\mu\Delta t + o(\Delta t^2) \end{aligned}$$

We finally get the following expression,

$$D_i(\underline{n},t) = \frac{n_i}{\bar{n}} P[s(t)=\underline{n}]$$

In the limit $t\to\infty$ we would have,

$$\begin{aligned} \lim_{t\to\infty} D_i(\underline{n},t) &= D_i(\underline{n}) \\ \lim_{t\to\infty} P[s(t)=\underline{n}] &= P(\underline{n}) \end{aligned}$$

where, $D_i(\underline{n})$ and $P(\underline{n})$ are the steady state probabilities. Thus, in the steady state we would have,

$$D_i(\underline{n}) = \frac{n_i}{\bar{n}} P(\underline{n}) \qquad \text{(A2)}$$

To find the average cost of a *find* operation we also have to derive the expression for the probability that a *find* observes the system in a given state at time t and the *find* is for a customer at a particular node. We denote the probability of a *find* encountering the system in state $\underline{n}$ at time t, where the *find* is for a resource at node i, by $F_i(\underline{n},t)$. We will follow a similar approach to that shown above for *delete* operations. Let now $A(t,t+\Delta t)$ stand for the event "A *find* occurs between times t and $t+\Delta t$", and let $B_i(t,t+\Delta t)$ stand for the event "A *find* for a resource at node i occurs between times t and $t+\Delta t$", then

$$F_i(\underline{n},t) = \lim_{\Delta t\to 0} P[s(t+\Delta t)=\underline{n}; B_i(t,t+\Delta t); s(t)=\underline{n}|A(t,t+\Delta t)]$$

After a derivation similar to the one performed for *delete*'s, we obtain,

$$\lim_{t\to\infty} F_i(\underline{n},t) = F_i(\underline{n}) = \frac{n_i}{\bar{n}} P(\underline{n}) \tag{A3}$$

Thus, $D_i(\underline{n}) = F_i(\underline{n})$, this, together with the fact that in the low load limit a resource is local if and only if its reference is maintained locally, implies that the average costs of both *delete*'s and *find*'s will be the same. We will thus derive only the cost for *delete* operations. The system is symmetric with respect to its nodes, thus the average cost for *delete* operations requested at node i will be the same as for node 1, thus in what follows we will consider only *delete* operations requested at node 1. Let $d(i,\underline{n})$ represent the cost of a *delete* operation requested at node 1 when the request is for a customer at node i and the system is found in state $\underline{n}$, then, taking into account that

$$d(i,\underline{n}) = \begin{cases} 0 & \text{if } i = 1 \\ 1 + \sum_{j=2}^{L} \delta(n_j) & \text{otherwise} \end{cases}$$

where,

$$\delta(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{otherwise} \end{cases}$$

Then we have,

$$\begin{aligned} C_d &= \sum_{\underline{n}} \sum_{i=1}^{L} d(i,\underline{n}) D_i(\underline{n}) \\ &= \sum_{\underline{n}} \sum_{i=2}^{L} (1 + \sum_{j=2}^{L} \delta(n_j)) D_i(\underline{n}) \\ &= \sum_{\underline{n}} \sum_{i=2}^{L} D_i(\underline{n}) + \sum_{j=2}^{L} \sum_{\underline{n}} \delta(n_j) \sum_{i=2} \frac{n_i}{\bar{n}} P(\underline{n}) \end{aligned} \tag{A4}$$

The first term in (A4) is the probability that a *find* be for a resource at a node other than node 1. Due to the symmetry of the system, the probability that a find be for a resource at node i is the same that it be for a resource at node j, which in turn has to be equal to $\frac{1}{L}$. Thus the first summation will reduce to

$$\sum_{\underline{n}} \sum_{i=2}^{L} D_i(\underline{n}) = \frac{L-1}{L} \tag{A5}$$

For the second term of (A4) we will use the fact that $P(\underline{n})$ has a product form. Let $\underline{n}^{-i}$ stand for the vector $\underline{n}$ in which the i^{th} component is missing, and let $\underline{n}^{-(i,j)}$ be defined similarly.

Let n^{-i} and $n^{-(i,j)}$ be the number of resources for vectors $\underline{n}^{-i}$ and $\underline{n}^{-(i,j)}$ respectively. Then the second summation above can be written as,

$$\begin{aligned} & \frac{1}{\bar{n}} \sum_{j=2}^{L} \sum_{n_1} P_1(n_1) \sum_{\underline{n}^{-1}} n^{-1} \delta(n_j) P(\underline{n}^{-1}) \\ = & \frac{1}{\bar{n}} \sum_{j=2}^{L} (\sum_{\underline{n}^{-1}} n^{-1} P(\underline{n}^{-1}) - e^{-\frac{\bar{n}}{L}} \sum_{\underline{n}^{-(1,j)}} n^{-(1,j)} P(\underline{n}^{-(1,j)}) \end{aligned}$$

but $\sum_{\underline{n}^{-1}} n^{-1} P(\underline{n}^{-1})$ is just the average number of resources in a system with $L-1$ nodes. And $\sum_{\underline{n}^{-(1,2)}} n^{-(1,2)} P(\underline{n}^{-(1,2)})$ is the average number of resources in a system with $L-2$ nodes. Thus the above two sums add up to $\frac{\gamma}{LK\mu}(L-1) = \bar{n}\frac{L-1}{L}$ and $\frac{\gamma}{LK\mu}(L-2) = \bar{n}\frac{L-2}{L}$, respectively. we would thus obtain,

$$\begin{aligned} C_d &= \frac{L-1}{L} + \frac{1}{\bar{n}} \sum_{j=2}^{L} (\bar{n}\frac{L-1}{L} - e^{-\frac{\bar{n}}{L}} \bar{n}\frac{L-2}{L}) \\ &= \frac{L-1}{L} + \frac{L-1}{L}((L-1) - (L-2)e^{-\frac{r}{KL}}) \end{aligned}$$

which can be rewritten as,

$$C_d = (2 + (L-2)(1 - e^{-\frac{r}{KL}}))\frac{L-1}{L} \tag{A6}$$

Since $F_i(\underline{n}) = D_i(\underline{n})$, we also have that

$$C_f = (2 + (L-2)(1 - e^{-\frac{r}{KL}}))\frac{L-1}{L} \tag{A7}$$

The average cost for all operations, without distinguishing between types, can be computed with the help of expressions in (3)

$$C = \frac{\mu}{\lambda + 2 \cdot \mu} \cdot C_d + \frac{\lambda}{\lambda + 2 \cdot \mu} \cdot C_f = \frac{\mu + \lambda}{\lambda + 2 \cdot \mu} C_d \tag{A8}$$

A.2 Heavy Load Limit

In this approximation we assume that all nodes have their multicast tables full at all times, the composition of the multicast tables does not change over time, all nodes have the same number of resources and the number of resources mapping to a given multicast address is the same for all such addresses. This approach will give good approximations for systems with a large number of resources in which simple additions or deletions of resources will

not affect the composition of the multicast tables. Each such possible configuration will be called a heavy load configuration. Given any system, there will always be a possible set of heavy load configurations that can be reached, and once one of them has been reached, the system will stay in it (if the load does not decrease). We will derive exact results for the average cost of the *delete* operations. For *add* and *find* operations we will be restricted to provide only upper and lowerbounds. These bounds, however, will be tight.

We note that the approximation will only be applicable for $K \geq M$ (the multicast tables have to be filled). One of the first things to notice is that if resource R has multicast address $\omega(R)$, was created at node i, and i belongs to the multicast group of $\omega(R)$, then the reference for R will have to be stored at i. To see how this will be true, consider that all resources created after the heavy load limit has been reached will have that property, and any old resource not fulfilling the property, will eventually be deleted, thus disappearing from the system.

A system's heavy load configuration will be characterized by the sets $\mathcal{N}(a)$, representing the set of nodes which contain multicast address a. We will also use the notation $A(n)$ to represent the set of multicast addresses in node n's multicast table. The cardinality of $A(n)$ will be M for all nodes. $N(a)$ will be used to denote the cardinality of $\mathcal{N}(a)$. $c(n,a)$ will denote the cost of an *add* operation for a resource which maps to multicast address a when the operation is requested at node n. Similarly, $d(n,a,m)$ and $f(n,a,m)$ will denote the costs of a *delete* and *find* operation, respectively, when the operation is requested at node n, for a resource which maps to multicast address a and is at node m. Under the current assumptions, it will be equally likely that an *add* be requested at any node and for any multicast address. Thus the probability that an *add* operation be requested at node n for an address a, is given by $P(n,a) = \frac{1}{LK}$. Also, under the current assumptions, it will be equally likely that a *find* or *delete* operation be requested at any node, be for any multicast address, and the resource which is the object of the operation be at any node. Thus the probability that a *find* (or *delete*) operation be requested at node n, be for multicast address a and affects a resource at node m, is given by $P(n,a,m) = \frac{1}{L^2K}$. We will first proceed to derive the cost for *add* operations. Let $N(\cdot)$ represent a possible heavy load configuration, then

$$E[\text{Cost of } add | N(\cdot)] = \sum_{n=1}^{L} \sum_{a=1}^{K} c(n,a) P(n,a)$$

Note that,

$$c(n,a) = \begin{cases} 0 & a \in A(n) \\ 2N(a)+1 & a \notin A(n) \end{cases}$$

thus,

$$\begin{aligned} E[\text{Cost of } add|N(\cdot)] &= \frac{1}{LK}\sum_{n=1}^{L}\sum_{a\notin A(n)}(2\cdot N(a)+1) \\ &= \frac{1}{LK}\sum_{a=1}^{K}\sum_{n\notin \mathcal{N}(a)}(2\cdot N(a)+1) \\ &= \frac{1}{LK}\sum_{a=1}^{K}(L-N(a))(2\cdot N(a)+1) \end{aligned}$$

ending in the following expression for the conditioned average cost:

$$E[\text{Cost of } add|N(\cdot)] = (2L-1)\frac{M}{K} - \frac{2}{LK}\sum_{a=1}^{K}N^2(a)+1 \tag{A9}$$

We will now provide upper and lower bounds for the average cost of *add* operations. To do that we will derive upper and lower bounds for the sum $\sum_a N^2(a)$. Because all multicast tables are full, the $N(a)$ will be constrained by $\sum_a N(a) = LM$. This implies that the lower bound will be reached for a configuration in which $|N(a) - N(b)| \leq 1$ for all pairs of addresses a, b. It can be seen that the above results in the following lower bound,

$$\sum_a N^2(a) \geq \lfloor\frac{LM}{K}\rfloor\left(2LM - K(\lfloor\frac{LM}{K}\rfloor+1)\right) + LM \tag{A10}$$

To upperbound $\sum_a N^2(a)$ we will have to maximize the number of multicast addresses with $N(a) = L$, then take one of the remaining multicast addresses and assigning it as many nodes as possible (which will be less than L), then assign only one node to the remaining multicast addresses. Thus, let q stand for the number of multicast addresses which have L nodes assigned to them. q will be given by

$$q = \lfloor\frac{LM-K}{L-1}\rfloor \tag{A11}$$

we can then have q addresses with $N(a) = L$, one address with $N(a) = LM - qL - K + q + 1$ and $K - q - 1$ addresses with $N(a) = 1$. Thus we can write the following upper bound for $\sum_a N^2(a)$,

$$\sum_a N^2(a) \leq qL^2 + (p+1)^2 + K - q - 1 \tag{A12}$$

where $p = LM - K - q(L-1)$.

Substituting (A10) into (A9) we obtain the following,

$$E[\text{Cost of } add|N(\cdot)] \leq C_a^u = (2L-1)\frac{M}{K} + 1 - \frac{2}{LK}\left[LM + \lfloor\frac{LM}{K}\rfloor\left(2LM - K(\lfloor\frac{LM}{K}\rfloor+1)\right)\right]$$

Thus, the average (over all configurations) of the *add* operation will have an upper bound, C_a^u, given by,

$$C_a = \sum_{N(\cdot)} E[\text{Cost of } add|N(\cdot)]P[N(\cdot)] \leq C_a^u \tag{A13}$$

Substituting (A12) into (A9) we would obtain a lowerbound for the average cost of *add* operations.

$$E[\text{Cost of } add|N(\cdot)] \geq C_a^l = (2L-1)\frac{M}{K} + 1 - \frac{2}{LK}(qL^2 + (p+1)^2 + K - q - 1)$$

Thus C_a^l would be the lowerbound for the cost of *add* operations averaged over all configurations.

$$C_a = \sum_{N(\cdot)} E[\text{Cost of } add|N(\cdot)]P[N(\cdot)] \geq C_a^l \tag{A14}$$

The average cost of a *delete* operation for a given configuration will be given by

$$E[\text{Cost of } delete|N(\cdot)] = \sum_{n=1}^{L}\sum_{a=1}^{K}\sum_{m=1}^{L} d(n,a,m)P(n,a,m)$$

The costs $d(n,a,m)$ will be given as follows,

$$d(n,a,m) = \begin{cases} 0 & a \in A(n) \text{ and } m = n & (A15a) \\ 0 & a \in A(n), m \notin \mathcal{N}(a) \text{ but the resource's reference is in } n & (A15b) \\ N(a) & a \in A(n), m \notin \mathcal{N}(a) \text{ the resource's reference is not in } n & (A15c) \\ N(a) & a \in A(n), m \in \mathcal{N}(a), m \neq n & (A15d) \\ N(a)+1 & a \notin A(n) & (A15e) \end{cases}$$

To properly count the contribution to the cost by (A15b) and (A15c) above, we have to note that the probability that one of m's resources mapping to multicast address a, where $m \notin \mathcal{N}(a)$, has its reference stored in $n \in \mathcal{N}(a)$ is $\frac{1}{N(a)}$, we can now write the expression for the average cost of *delete* operations,

$$E[\text{Cost of } delete|N(\cdot)] = \frac{1}{L^2K}\sum_{n=1}^{L}\left(\sum_{a\in A(n)}\left(\sum_{\substack{m\in\mathcal{N}(a)\\ m\neq n}} N(a) + \sum_{m\notin\mathcal{N}(a)} \frac{N(a)-1}{N(a)}N(a)\right) + \sum_{a\notin A(n)}\sum_{m=1}^{L}(N(a)+1)\right)$$

$$
\begin{aligned}
&= \frac{1}{LK}\sum_{a=1}^{K}\left(\sum_{n\in\mathcal{N}(a)}(N(a)-1)+\sum_{n\notin\mathcal{N}(a)}(N(a)+1)\right)\\
&= \frac{1}{LK}\sum_{a=1}^{K}(N(a)(N(a)-1)+(L-N(a))(N(a)+1))\\
&= (L-2)\frac{M}{K}+1
\end{aligned}
$$

Thus, $E[\text{Cost of } \textit{delete}|N(\cdot)]$ does not depend on the particular configuration, which allows us to write,

$$C_d = (L-2)\frac{M}{K}+1 \tag{A16}$$

The average cost of find operations conditioned to a particular configuration is given by,

$$E[\text{Cost of } \textit{find}|N(\cdot)] = \sum_{n=1}^{L}\sum_{a=1}^{K}\sum_{m=1}^{L} f(n,a,m)P(n,a,m)$$

Where the costs $f(n,a,m)$ will be given by

$$
f(n,a,m) = \begin{cases}
0 & m = n & (A17a)\\
0 & a \in A(n), m \notin \mathcal{N}(a), \text{ the resource's reference is in } n & (A17b)\\
N(a) & a \in A(n), m \notin \mathcal{N}(a), \text{ the resource's reference is not in } n & (A17c)\\
N(a) & a \in A(n), m \in \mathcal{N}(a), m \neq n & (A17d)\\
N(a)+1 & a \notin A(n), m \neq n & (A17e)
\end{cases}
$$

The same remark applies now to properly compute the contribution to the cost of (A17b) and (A17c).

$$
\begin{aligned}
E[\text{Cost of } \textit{find}|N(\cdot)] &= \frac{1}{L^2K}\sum_{n=1}^{L}\left(\sum_{a\in A(n)}\left(\sum_{\substack{m\in\mathcal{N}(a)\\ m\neq n}}N(a)+\sum_{m\notin\mathcal{N}(a)}\frac{N(a)-1}{N(a)}N(a)\right)\right.\\
&\quad\left.+\sum_{a\notin A(n)}\sum_{\substack{m=1\\ m\neq n}}^{L}(N(a)+1)\right)
\end{aligned}
$$

Thus we finally get

$$E[\text{Cost of } \textit{find}|N(\cdot)] = (L-1)\frac{K+M(L-2)}{LK} - \frac{M}{LK} + \frac{1}{L^2K}\sum_{a=1}^{K}N^2(a) \tag{A18}$$

Substituting equation (A12) in (A18) we get an upper bound for $E[\text{Cost of } \textit{find} | N(\cdot)]$, C_f^u, which will also be an upper bound for the cost of *find* averaged over all possible heavy load configurations. Thus,

$$C_f \leq C_f^u = (L-1)\frac{K+M(L-2)}{LK} - \frac{M}{LK} + \frac{1}{L^2K}(qL^2 + (p+1)^2 + K - q - 1) \quad \text{(A19)}$$

Substituting (A10) into equation (A18) we obtain a lower bound for C_f, C_f^l,

$$\begin{aligned} C_f \geq C_f^l &= (L-1)\frac{K+M(L-2)}{LK} - \frac{M}{LK} \\ &+ \frac{1}{L^2K}\left[LM + \lfloor\frac{LM}{K}\rfloor\left(2LM - K(\lfloor\frac{LM}{K}\rfloor + 1)\right)\right] \end{aligned} \quad \text{(A20)}$$

The average cost per operation without distinguishing operation classes would be given by

$$C = \frac{\lambda}{\lambda+2\mu}C_f + \frac{\mu}{\lambda+2\mu}(C_a + C_d) \quad \text{(A21)}$$

With the proper substitutions, (A21) provides us with the corresponding upper and lower bounds for the overall average costs, C^u and C^l, respectively.

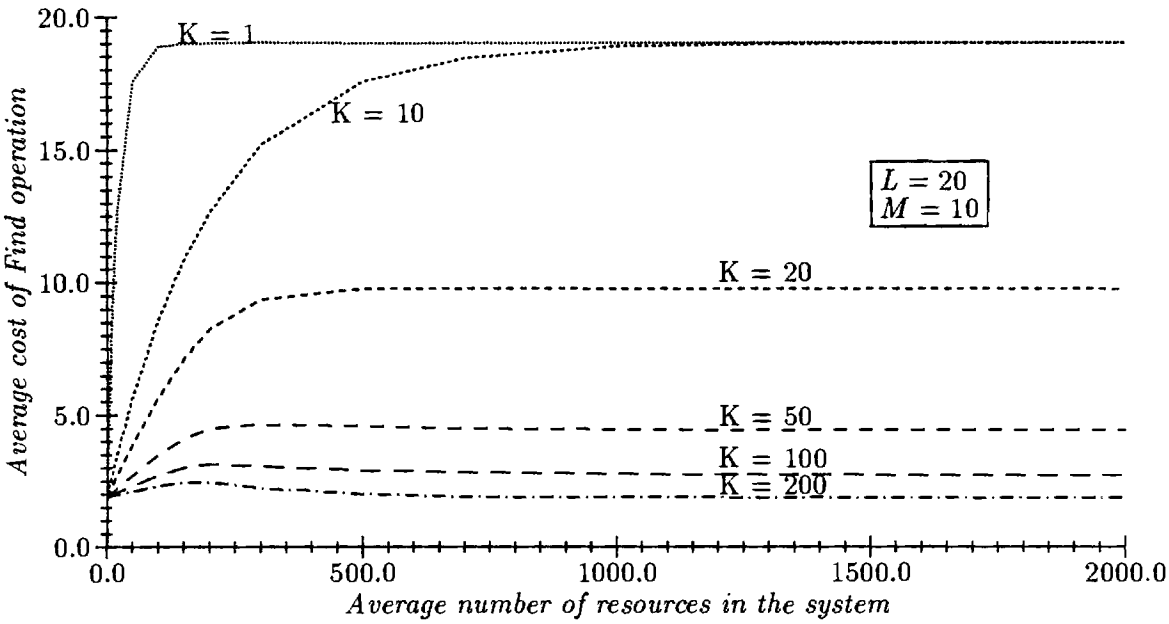


Figure 1: Average cost of *find* versus average number of resources

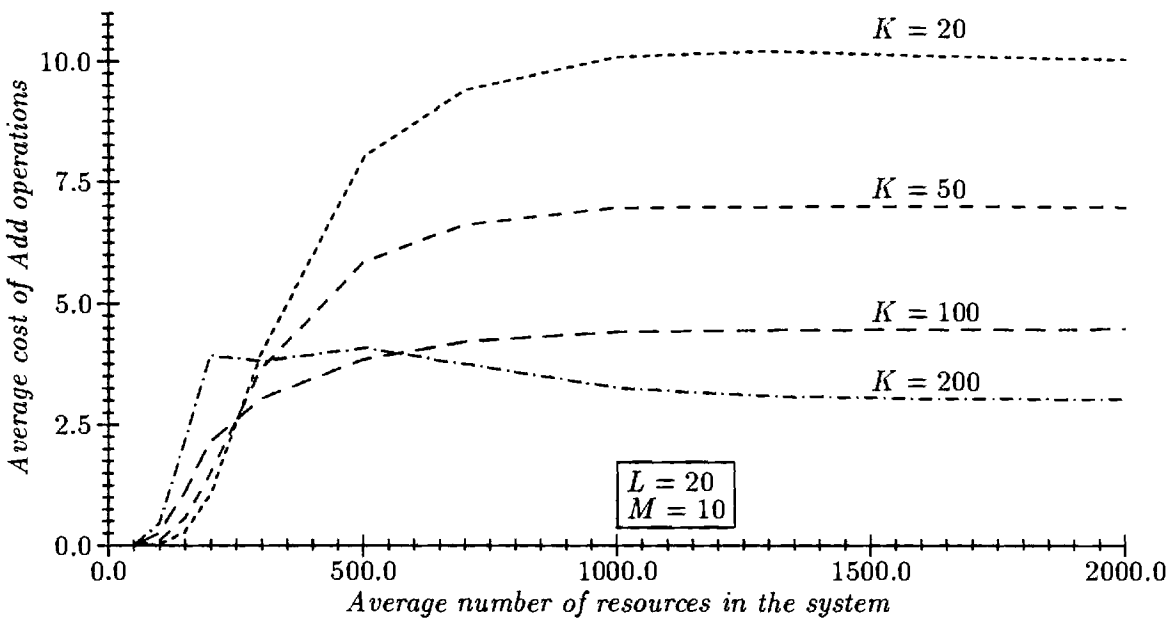


Figure 2: Average cost of *add* versus average number of resources

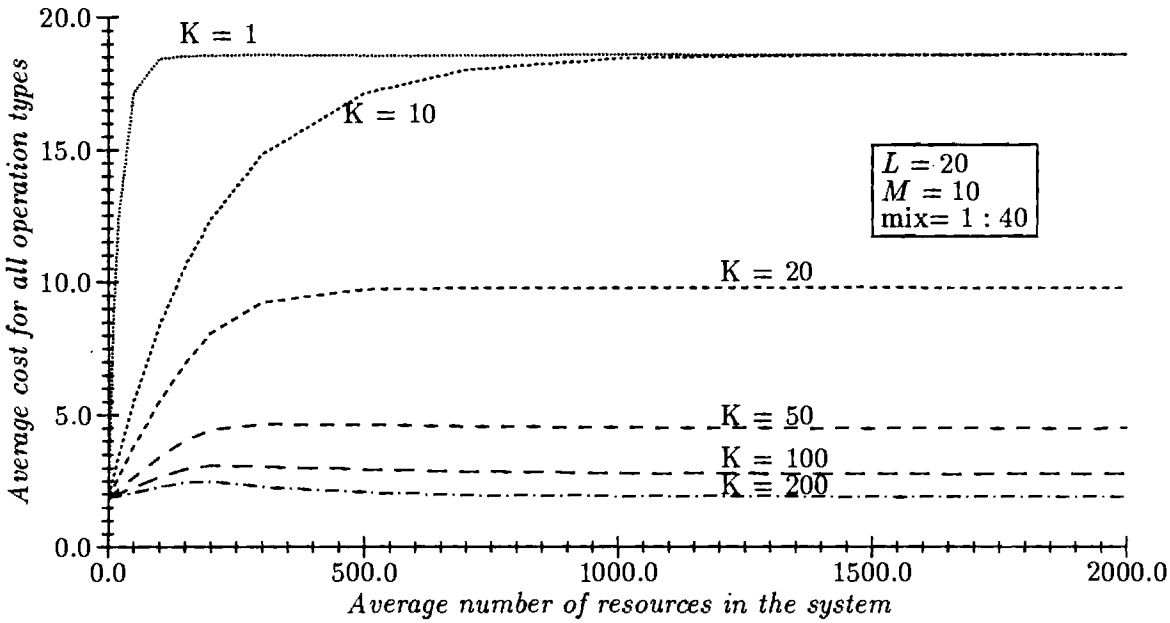


Figure 3: Average overall cost of versus average number of resources

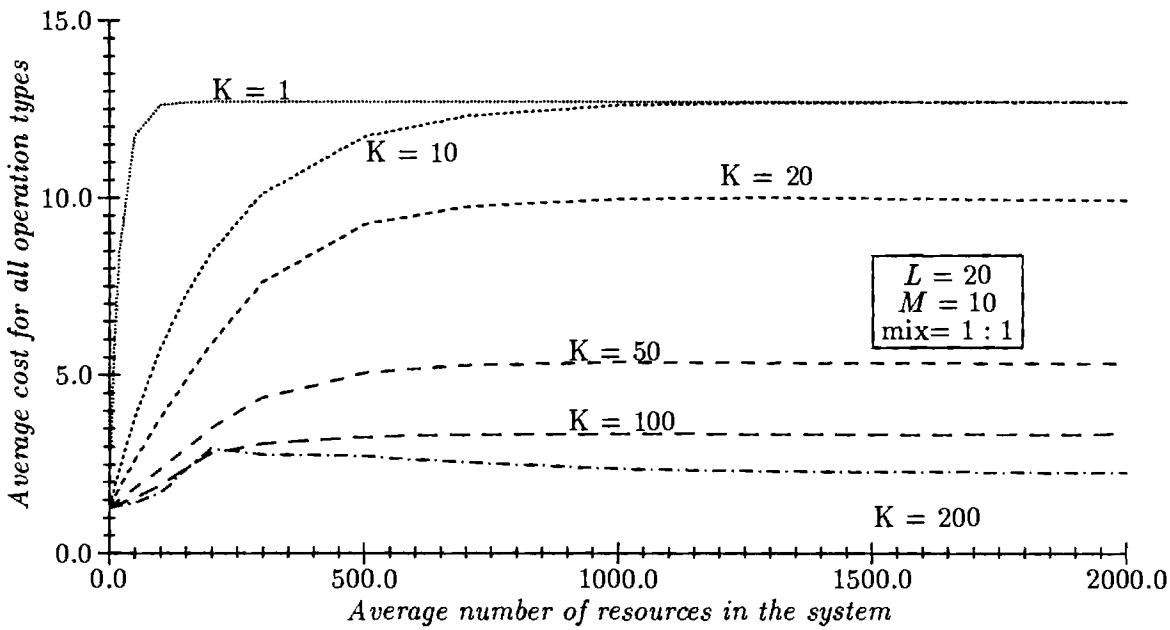


Figure 4: Average overall costs versus average number of resources

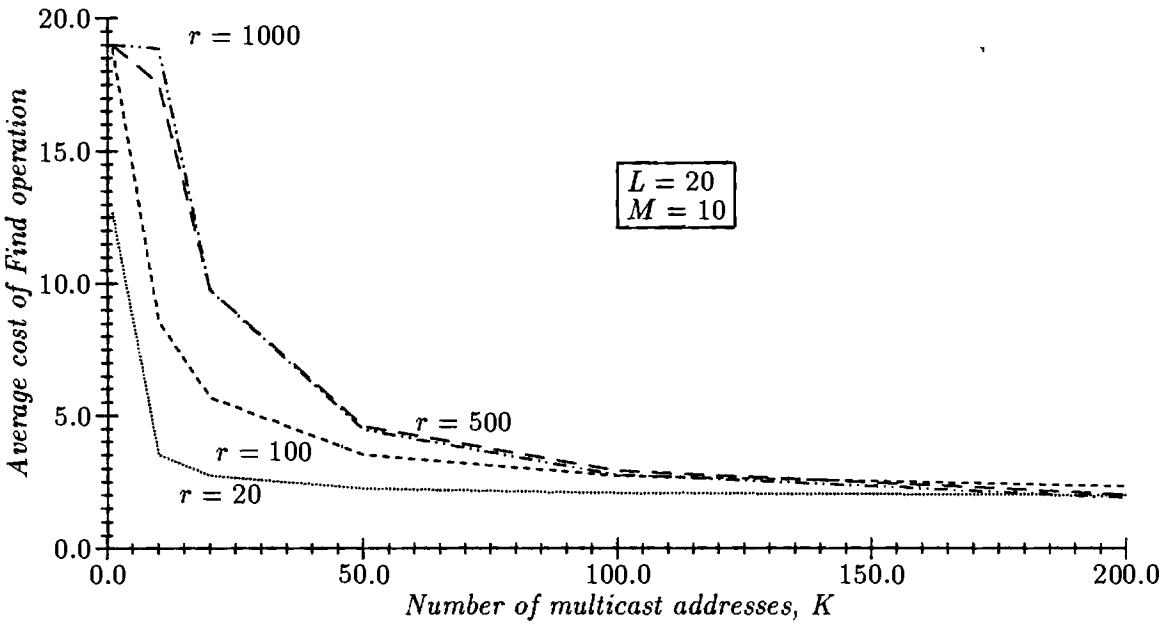


Figure 5: Average cost of *find* versus number of addresses

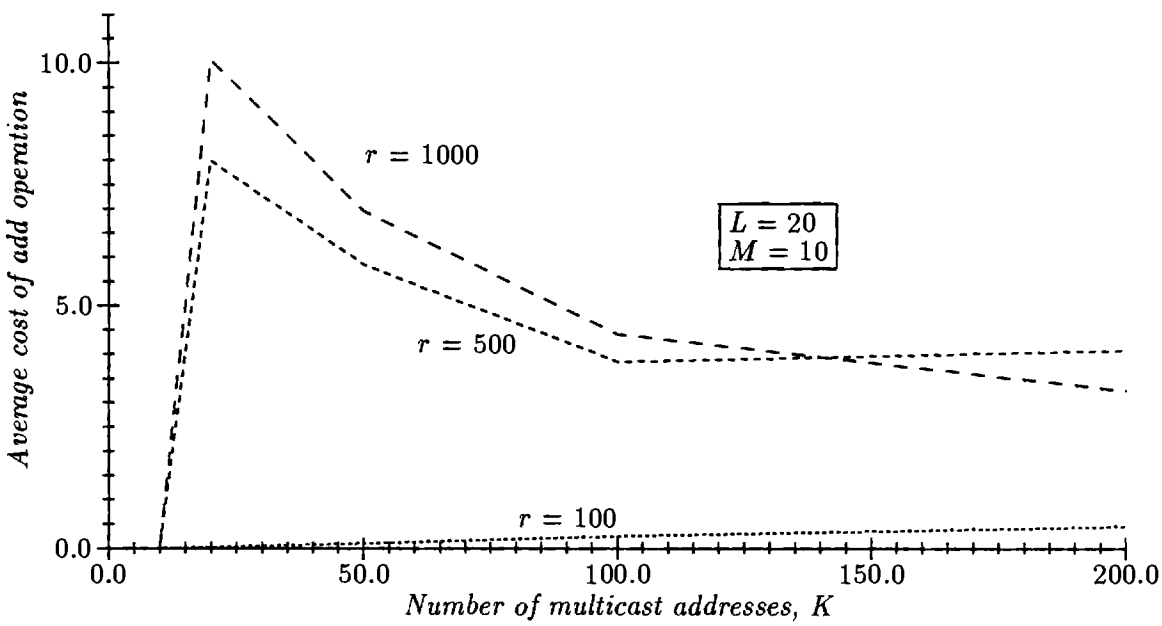


Figure 6: Average cost of *add* versus number of addresses

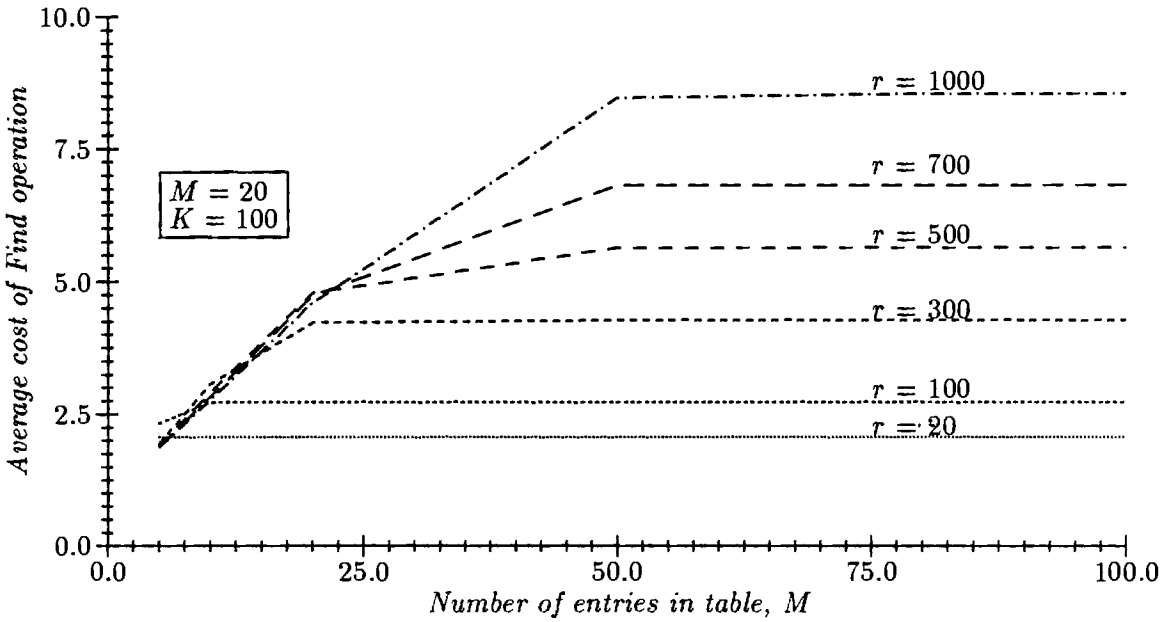


Figure 7: Average cost of *find* versus number of entries

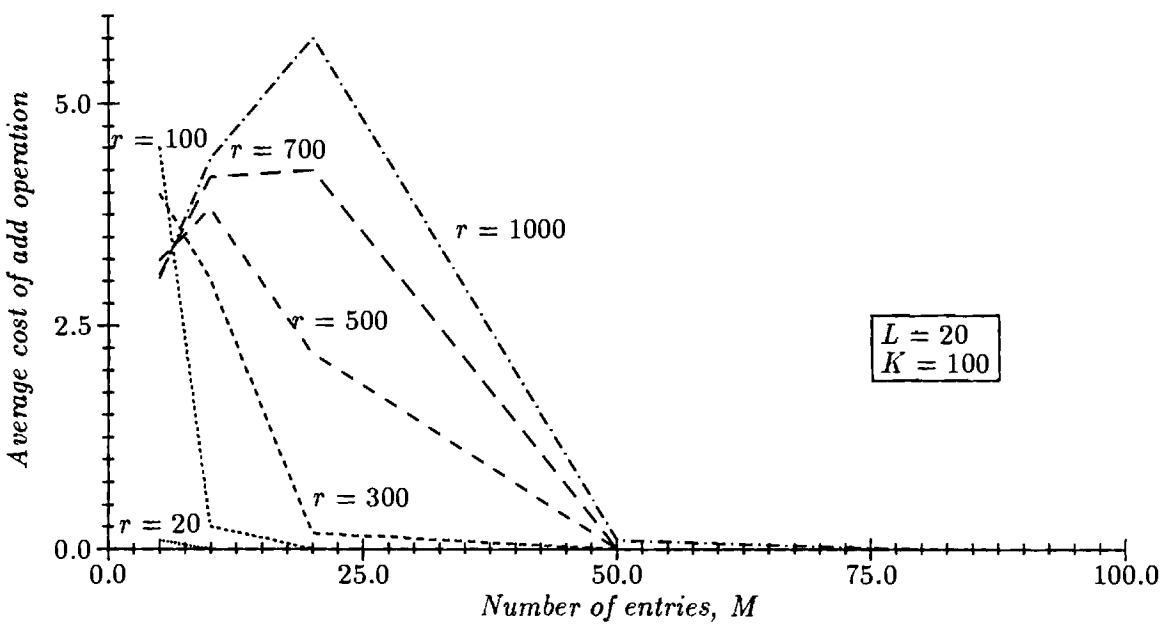


Figure 8: Average cost of *add* versus number of entries

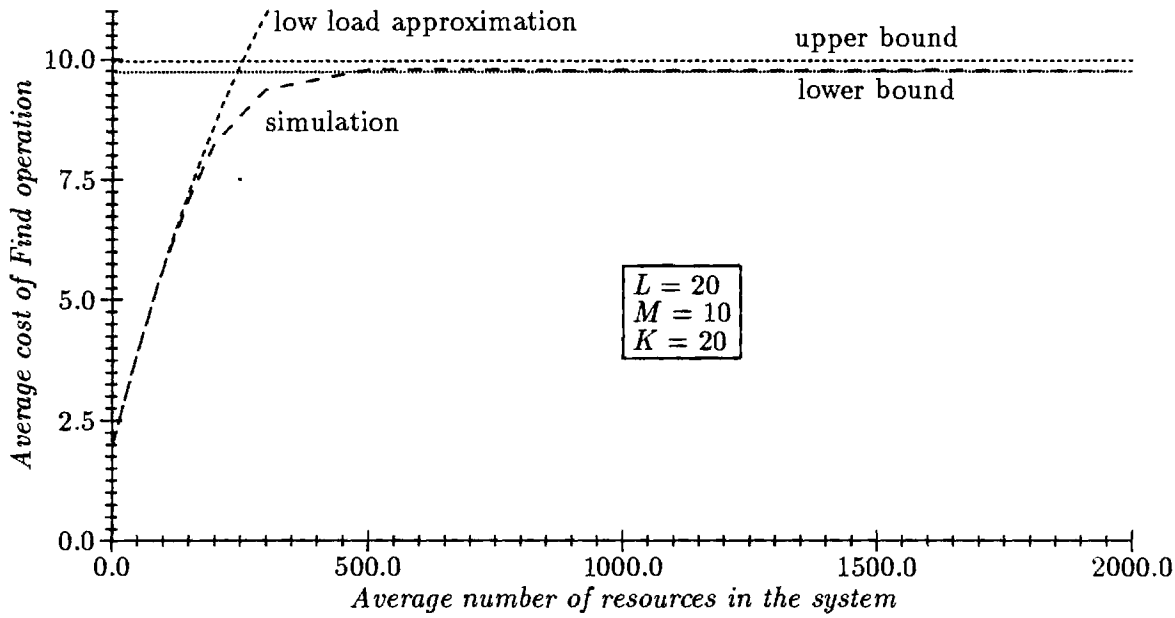


Figure 9: Comparison of analysis with simulation

References

[ABA88] M. Ammar, J. Bernabeu, and M. Ahamad. Using Hint Tables to Locate Resources in Distributed Systems. In *INFOCOM*, IEEE, 1988.

[BCMP75] F. Baskett, K. Chandy, R. Muntz, and R. Palacios. Open, closed and mixed networks of queues with different classes of customers. *Journal of the ACM*, April 1975.

[BLNS82] A. D. Birrel, R. Levin, R. M. Needham, and M. D. Schroeder. Grapevine: an exercise in distributed computing. *Communications of the ACM*, 25(4):260–274, April 1982.

[CM86] D. R. Cheriton and T. P. Mann. *A Decentralized Naming Facility.* Technical Report STAN–CS–1098, Department of Computer Science, Stanford University, February 1986. Revised version to appear in ACM Transactions on Computer Systems.

[Deu83] *Deuna Users Manual.* Digital Equipment Corporation, 1983.

[DLS85] P. Dasgupta, R. J. LeBlanc, and E. Spafford. *The Clouds Project: Design and Implementation of a Fault-Tolerant Distributed Operating System.* Technical Report GIT-ICS-85/29, Georgia Institute of Technology, 1985.

[Fow85] R.J. Fowler. *Decentralized Object Finding Using Forwarding Addresses.* PhD Thesis 85-12-1, University of Washington, December 1985.

[Kle75] L. Kleinrock. *Queuing Systems.* Volume 1: *Theory*, John Wiley & Sons, 1975.

[Lit61] J.D.C. Little. A Proof of the Queueing Formula $L = \lambda W$. *Operations Research*, 9:383–387, 1961.

[MV85] S.J. Mullender and P.M.B. Vitányi. Distributed match–making for processes in computer networks. In *Fourth ACM Symposium on the Principles of Distributed Computing*, ACM, Minacki, Ontario, August 1985.

[OD83] D.C. Oppen and Y.K. Dalal. The clearinghouse: a decentralized agent for locating named objects in a distributed environment. *ACM Transactions on Office Information Systems*, 1(3):230–253, July 1983.

[Ter87] D.B. Terry. Caching hints in distributed systems. *IEEE Transactions on Software Engineering*, SE-13(1):48–54, January 1987.

Optimal Selection of Multicast Groups for Resource Location in a Distributed System*

José M. Bernabéu-Aubán
Mostafa H. Ammar
Mustaque Ahamad

Technical Report GIT-ICS-88/27

August 1, 1988

Abstract

In this paper we present a protocol to locate (or find) named resources in a distributed system which uses the multicast capabilities of the underlying network. Each node in the network uses a sequence of node groups, and each node group is associated with a unique multicast address. To locate a resource, the searching node sequentially polls each one of the groups until the resource is found. This scheme is a generalization of both pure polling and broadcast. Our basic aim is to show how to obtain an optimal division of the nodes into multicast groups. To that end, the protocol is analyzed and an efficient algorithm is given that provides a group division minimizing the expected cost per location operation.

School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

*This work was supported in part by NSF grants NCR-8604850, CCR-8806358 and CCR-8619886.

1 Introduction

Distributed systems offer many advantages over centralized ones, including fault–tolerance, resource sharing and increased parallelism. The task of programming a distributed system, however, is more difficult because the global system state is not available. In particular, in systems where resources (e.g., files, processes) can be migrated between nodes, it would be necessary for the user to implement a procedure to find resources needed by the computation. To avoid this, the system must offer the users the abstraction of a unified system in which the location of resources is transparent to them. In such a system, the Operating System should implement an algorithm to find the location of a remote resource. When a resource may be used repeatedly at a node, caching its address locally [1,3,4,5], is a widely used technique for reducing the cost of determining the location of a resource. Since the cache information may be incorrect, the general problem of finding the resource still exists when caching is used.

A widely used scheme to find resources involves the use of name servers [1,2]. In its simplest form, one of the nodes in the network is designated as the name server for the whole system. When a node needs to locate a resource, it directs a request to the name server. When a resource moves between nodes, an update message is sent to notify the name server. In a large system, such a name server would become a bottleneck, degrading the performance of the system. Also, the single name server approach would be especially vulnerable to node failures. A more general approach can distribute the name server task among several nodes and, a particular name server usually takes care of only a part of the resource name space. The problem now is to decide which name server to contact to find the location of a resource. A resource's location may now be found by broadcasting (actually multicasting [7,8]) to all name servers requesting that they provide the resource's address. Another approach, used in the R* system [3], is to encode the name of the node where a resource was created in the resource's name. Then that node will function as the resource's name server.

In the absence of name servers, a node wishing to determine the location of a resource, can send a broadcast message to all nodes and make them search their local directories. This is the approach taken in the *Clouds* operating system [6]. Broadcasting, though simple, would waste computational resources at every node, where it would compete with the local computations for CPU time. For large rates of location requests this would rapidly degrade the performance of the entire system.

At the other extreme, if the individual nodes are polled sequentially, this would certainly

decrease the amount of CPU time wasted in the system (especially if the nodes more likely to know about the resource were consulted first). However this approach would also increase the bandwidth utilization, because many messages will be sent. Since the messages are sent sequentially, the real disadvantage of this approach is that the location operations would take longer (larger response time).

In this paper we present a location protocol which considers a cost measure that includes both the CPU utilization and the response time. The approach taken is based on a scheme in which the nodes in the network are divided into disjoint multicast groups, and are polled by a sequence of multicast messages. The two approaches mentioned above are just special cases when all nodes are reached by a single message (broadcast) or when only one node is reached with each message (polling). We also present a cost model for the system and an efficient algorithm which, based on the probability distribution of a resource's location among the nodes in the network, finds the optimal decomposition into disjoint groups, as well as the optimal sequence in which the groups should be polled.

In section 2 we give a description of the protocol operation. In section 3 we present the model of the system to be used for the cost analysis carried out in section 4. Section 5 describes an algorithm to determine an optimal multicast grouping. Some numerical examples are presented in section 6. In section 7 we discuss some practical considerations in the assignment of multicast addresses. Finally in section 8 we present some concluding remarks.

2 Protocol Description

We now describe the operation of the Multicast Location Protocol (MLP) which is invoked whenever a node needs to locate a resource that is not found locally. MLP is intended to be used in a bus-based network with broadcast and multicast capabilities. The protocol, with minor modifications may also be applied to other network topologies such as Ring LAN's and store–and–forward Packet–Switched networks. MLP assumes that there is a subset of the nodes, $\mathcal{N}$, that can potentially hold the resource. This set is called the *authoritative set* and is assumed to be divided into multicast groups $(g_1, \ldots, g_K)$ such that $g_i \cap g_j = \phi$ and $\cup_{i=1}^{K} g_i = \mathcal{N}$. (Determining an optimal sequence of multicast groups is the topic discussed in section 5). Before the scheme proposed is used, it is assumed that the multicast addresses for all the groups needed have been distributed to each node in the multicast groups, and the nodes have installed them in their network interfaces (see section 7). In the operation of the protocol it is assumed that at most one of the authoritative nodes knows about a

particular resource at any particular moment.

In searching for the resource, the node goes through a set of at most $K + 1$ phases. The search is terminated if the resource is found in any particular phase. During phase i, $i = 1, \ldots, K$, the searching node attempts to locate the resource by multicasting to the nodes in the group g_i. The node proceeds to phase $i + 1$ after it determines that it is unlikely that the resource is contained in group g_i. As will be explained later, and because of transmission errors, it is possible (although somewhat unlikely) that the node will fail in locating the resource in a group that contains it. Such an eventuality is handled by phase $K + 1$, called the broadcast phase, in which all nodes in the authoritative set are searched simultaneously.

Phase $i, i = 1, \ldots, K$, begins when the searching node is ready to transmit a *multicast location (ML) message* to the nodes in group g_i. Because of possible contention on the transmission channel, the node will require some time until it can begin a successful transmission of the message (see figure 1.) All nodes in the network that hear a correctly transmitted multicast location message will refrain from transmitting any packets for a period of time Δ. The only exception to this rule is any node in g_i that contains the resource identified in the message (there is at most one). If such a node exists, it responds by starting a *location response (LR) message* in the time period Δ.

Assuming no errors on the channel, each phase may have one of two outcomes, as shown in figure 1,

1. The searching node hears nothing during the period Δ and will thus proceed to phase $i + 1$.
2. The searching node hears the beginning of a LR message during Δ. The search is thus terminated at the end of the message reception.

If errors occur in the transmission of the ML message, nodes in the network will not refrain from transmission during the time period Δ and thus the searching node may hear unrelated (successful and correct or collided) traffic during that period. In this case the node knows that its message was in error and thus remains in phase i and gets ready to retransmit the ML message to group g_i (this is shown in figure 2). It is also possible that, despite errors in the ML message, the searching node hears nothing during the period Δ. The node cannot tell that its message was not received correctly and will thus go on to start phase $i + 1$. It is because of this last situation that the searching node may miss locating the resource in the proper phase, and we thus require the broadcast phase (phase $K + 1$).

Errors in the LR message will cause the searching node to hear something in response to its ML message during the period Δ. However what is heard is undecipherable. Thus the searching node will assume somebody is trying to respond and will remain in phase i, getting ready to retransmit the ML message to group g_i (see figure 2).

In general, thus, each phase will consist of several subphases, which we will call *intervals.* Each interval begins when the node is ready to transmit a ML message and ends when the node is ready to send a new or retransmitted ML message, or when the resource is found.

If at the end of the first K phases the resource has not been located, the searching node will enter the broadcast phase, phase $K+1$, which is similar to the other phases with two exceptions,

1. The message sent by the searching node is a *broadcast location (BL) message*, addressed to all nodes in the authoritative set.
2. The phase ends only when the resource has been found or when an upper bound on the allowed number of retransmissions of the broadcast location message is reached, whichever comes first. In this latter case an error condition is reported to the application process searching for the resource.

3 Model of the System

We consider a given node searching for a resource in an authoritative set $\mathcal{N}$ of size N, using MLP. The sequence of multicast groups, $G = (g_1, \ldots, g_K)$, is given and is assumed to have the properties discussed in section 2. Such a sequence of multicast groups will be referred to as a *K-search sequence defined over $\mathcal{N}$*. The number of nodes in g_i is denoted by n_i.

Our primary interest in the analysis to follow is the determination of an optimal sequence of multicast groups. We optimize relative to a performance measure that is a combination of two costs:

1. Search cost: This is the cost incurred whenever a node receives a ML message addressed to a group in which it is a member. This causes the node to interrupt its current processing and search its local resource directory to determine whether it contains the resource. We assume that this cost is the same for all nodes in the network, and we take it as our unit of CPU cost.
2. Delay cost: This represents the amount of time spent by the searching node from the beginning of phase 1 until the end of the last phase.

In general, the costs above will depend on the particular sequence, G, of multicast groups. We will thus denote the average search and delay costs by $S(G)$ and $D(G)$ respectively. The overall performance measure we consider is the weighted sum:

$$C(G) = \alpha S(G) + \beta D(G) \tag{1}$$

Where α and β are real values whose ratio indicates the relative importance of each cost measure.

In calculating the above costs we will also assume the following

1. The average length of each interval is given by t. This length incorporates the time needed until the completion of a successful transmission of a ML message plus the time until the node is ready to transmit the next ML message (see figure 2). In general, t will depend on the particulars of the media access protocol used, and the offered load the network is experiencing. The value for t may be calculated using available analysis (see, e.g., [9] for CSMA/CD) or measured if a real system is available.

2. The probability that a ML, BL or LR message will be transmitted in error is given by q. (We will need to assume that $q \leq 1/3$ in the proof of Lemma 2 in section 5; not an unrealistic assumption.) We assume that messages in error are not received correctly by any node.

3. When a ML message is in error, the probability that some other node in the network tries to transmit a message during the period Δ is given by σ.

4. There is no limit on the number of retransmissions allowed for the BL message during the broadcast phase.

Finally, we assume the existence of an *a priori* probability distribution, P_a for $a \in \mathcal{N}$, such that $\sum_{a \in \mathcal{N}} P_a = 1$. P_a will denote the probability that the resource will be found in node a. We also use $P(g_i)$ to denote the *a priori* probability that the resource will be found in group g_i. We thus have that $P(g_i) = \sum_{a \in g_i} P_a$. The special case where $P_a = \frac{1}{N}$ for all $a \in \mathcal{N}$, models the situation where the searching node only knows the authoritative set but has no other information regarding the resource's whereabouts.

4 Analysis

4.1 The Cost of a Search Sequence

Based on the model of the system presented in the previous section we will now derive an expression for the cost associated with a given K-search sequence. To find $C(G)$, we use the following expression,

$$C(G) = \sum_{i=1}^{K} P(g_i) E[\text{ Cost of MLP } | \text{ resource in group } g_i] \tag{2}$$

Assuming that the resource is in group g_i, the searcher will go through phases 1 to $i-1$ without finding the resource, and the cost incurred by this fruitless search will be

$$\sum_{j=1}^{i-1} E[\text{ Cost of multicast phase } j \mid \text{ resource not in } g_j \,] = \sum_{j=1}^{i-1} C_{1j} \tag{3}$$

The node will also incur a cost in performing phase i, this cost will be denoted by

$$E[\text{ Cost of multicast phase } i \mid g_i \text{ contains the resource }] = C_{2i} \tag{4}$$

Furthermore, there is a possibility that the resource is not found in phase i even though the resource is in group g_i. In such case, the searcher will proceed through phases $i+1$ through K, incurring the following cost,

$$\sum_{j=i+1}^{K} E[\text{ Cost of multicast phase } j \mid \text{ given that the resource is not in group } g_j] = \sum_{j=i+1}^{K} C_{1j} \tag{5}$$

Finally, if the resource is not found during phase i, it can only be found during the broadcast phase. The extra cost incurred when the resource is missed in phase i, due to the execution of the broadcast phase is

$$E[\text{ Cost of broadcast phase }] = C_B \tag{6}$$

Summarizing, the cost of a search sequence can be expressed in the following way:

$$\begin{aligned} C(G) \;=\; & \sum_{i=1}^{K} P(g_i) \left\{ \sum_{j=1}^{i-1} C_{1j} + C_{2i} \right. \\ & + \left. Pr[\text{ resource not found in phase } i \mid \text{ resource in } g_i] \cdot \left[C_B + \sum_{j=i+1}^{K} C_{1j} \right] \right\} \end{aligned} \tag{7}$$

4.2 Cost of a Phase for Groups not Containing the Resource

First note that according to MLP, a phase exploring a group that does not contain the resource will only terminate when activity is detected during the period Δ. Thus a phase consisting of M intervals will have to have $M-1$ intervals at the beginning in which the ML message was in error and some nodes transmitted during the period Δ, plus one interval at the end in which no transmission was detected during the period Δ. The probability that in an interval the ML message is in error and some node in the system uses the channel during the period Δ is $q\sigma$. On the other hand, in the last interval, the silence during the period Δ can be due to two reasons:

1. The ML message was not in error. This will happen with probability $(1-q)$.
2. The ML message was in error, but no node used the channel during the period Δ. This will happen with probability $q(1-\sigma)$.

Thus the probability that the phase consists of M intervals and the last one sent an error-free ML message is given by $(q\sigma)^{M-1}(1-q)$. The search cost of phase j in this case would be αn_j (the nodes in the group perform local searches only during the last interval in which they receive an error–free ML message). The phase would, however, incur a delay penalty for every interval. Thus the delay cost would be given by $M\beta t$. The probability that the phase consisted of M intervals and the last one had the ML message in error is, $(q\sigma)^{M-1}q(1-\sigma)$. In this case the search cost of the phase is zero (no node performs local searches), and the delay cost is $M\beta t$. Thus the expected cost for phase j is given by,

$$\begin{aligned} C_{1j} &= \sum_{M=1}^{\infty} \left\{ (q\sigma)^{M-1}(1-q)(\alpha n_j + M\beta t) + (q\sigma)^{M-1}q(1-\sigma)M\beta t \right\} \\ &= \frac{1}{1-q\sigma}((1-q)\alpha n_j + \beta t) \end{aligned} \tag{8}$$

4.3 Cost of Phase for Group Containing the Resource

We now find the expected cost for multicast phase i when g_i contains the resource, C_{2i}. A phase consisting of M intervals will, in general, have m of them for which the ML message is error–free and ℓ for which the ML message is in error, with $M = m + \ell$. In this case, an interval sending an error–free ML message will not be the last one in the phase, only if the RL message is in error. This can happen with probability $(1-q)q$. Also, an interval will send an ML message in error and be the last interval in the phase only if some other node

made use of the channel during the period Δ. This will happen with probability $q\sigma$. Thus an interval will be the last interval of the phase in either of two cases:

1. Both the ML and RL messages were error–free. In this case the resource will be found in the phase. This will have a probability of $(1-q)^2$
2. The ML message was in error, but no other node in the network used the channel during period Δ. This will happen with probability $q(1-\sigma)$. In this case the resource will not be found in the phase.

Let us denote by $Q_F(m,\ell)$ the probability that the phase takes m intervals with error-free ML messages, and ℓ intervals with ML messages in error, finally finding the resource. In this case, the last interval has an error-free ML message. There are, on the other hand, $\binom{(m-1)+\ell}{\ell}$ ways of choosing the positions of the ℓ messages with an erroneous ML message. Thus we obtain the following,

$$Q_F(m,\ell) = \binom{m-1+\ell}{\ell}((1-q)q)^{m-1}(q\sigma)^{\ell}(1-q)^2 \tag{9}$$

Let us now denote by $Q_{NF}(m,\ell)$, the probability that, the phase contains m intervals with error–free ML messages and ℓ intervals with ML messages in error, but with the last interval not finding the resource. In this case, the last interval has to have an ML message in error. Then we obtain the following,

$$Q_{NF}(m,\ell) = \binom{m+\ell-1}{\ell-1}((1-q)q)^{m}(q\sigma)^{\ell-1}q(1-\sigma) \tag{10}$$

From the above expressions we can, by summing over m and ℓ, compute the marginal probabilities $Q_F \equiv$ Prob[resource is found in phase i | resource in g_i]. and $Q_{NF} \equiv$ Prob[resource not found in phase i | resource in g_i].

$$Q_F = \frac{(1-q)^2}{(1-q)^2+q(1-\sigma)} \tag{11}$$

$$Q_{NF} = \frac{q(1-\sigma)}{(1-q)^2+q(1-\sigma)} \tag{12}$$

Using equations (9), and (10), we can find the average number of intervals with error–free ML messages for each type of outcome(R_F and R_{NF}), obtaining

$$\begin{aligned} R_F &= \text{average number of intervals such that the ML message} \\ &\quad \text{is error–free, and the resource is found at the end} \\ &= \sum_{m=1}^{\infty}\sum_{\ell=0}^{\infty} mQ_F(m,\ell) = \frac{(1-q)^2(1-q\sigma)}{((1-q)^2+q(1-\sigma))^2} \end{aligned} \tag{13}$$

Similarly,

$$\begin{aligned} R_{NF} &= \text{average number of intervals such that the ML message} \\ &\quad \text{is in error, and the resource is not found at the end} \\ &= \sum_{\ell=1}^{\infty}\sum_{m=0}^{\infty} mQ_{NF}(m,\ell) = \frac{q(1-\sigma)q(1-q)}{((1-q)^2+q(1-\sigma))^2} \end{aligned} \tag{14}$$

The average number of intervals, R, being given by

$$\begin{aligned} R &= \sum_{m=0}^{\infty}\sum_{\ell=0}^{\infty}(m+\ell)(Q_F(m,\ell)+Q_{NF}(m,\ell)) \\ &= \frac{1}{(1-q)^2+q(1-\sigma)} \end{aligned} \tag{15}$$

The cost C_{2i} would be given by

$$\begin{aligned} C_{2i} &= (R_F+R_{NF})\alpha n_i + R\beta t \\ &= \frac{1}{(1-q)^2+q(1-\sigma)}[(1-q)\alpha n_i+\beta t] \end{aligned} \tag{16}$$

4.4 Cost of the Broadcast Phase

Finally we find the expected cost of the broadcast phase. We first note that, according to our assumption, this phase will only terminate when the resource has been found, which implies that in the last interval of the phase, the ML and RL messages are error–free. As for a phase in which the group contains the resource, the broadcast phase will consist of $m+\ell$ intervals, with m intervals in which the ML message is error–free (the last one is the final interval, during which the resource is found), and ℓ intervals in which the ML message is in error. The probability that an interval has the ML message in error is q, whereas the probability that an interval has an error–free ML message, while the RL message is in error is given by $(1-q)q$. Finally, the probability that an interval has both ML and RL messages error–free is $(1-q)^2$. There are $\binom{(m-1)+\ell}{\ell}$ ways of choosing the position of the ℓ intervals with erroneous ML messages in the broadcast phase. We use $Q(m,\ell)$ to denote the probability that the phase is composed of $m+\ell$ parts, m of which have an error–free ML message, and ℓ of which have a ML message in error. $Q(m,\ell)$ would be given by:

$$Q(m,\ell) = \binom{m-1+\ell}{\ell}((1-q)q)^{m-1}q^{\ell}(1-q)^2 \tag{17}$$

Thus, in a broadcast phase there would be m intervals in which the nodes would perform local searches. Thus the search cost of the phase would be given by $m\alpha N$. A delay penalty

is incurred for every phase. Thus the delay cost is given by $(m+\ell)\beta t$. This leads to the following expression for the average cost,

$$\begin{aligned} C_B &= \sum_{m=1}^{\infty}\sum_{\ell=0}^{\infty} Q(m,\ell)(m\alpha N + (m+\ell)\beta t) \\ &= \frac{1}{(1-q)^2}((1-q)\alpha N + \beta t) \end{aligned} \tag{18}$$

4.5 Average Cost of a Search Sequence

At this point we have all we need to compute $C(G)$ from equations (7),(8), (11), (12) and (16).

$$\begin{aligned} C(G) = \sum_{i=1}^{K} P(g_i) \Bigg\{ & \frac{1}{1-q\sigma}\sum_{j=1}^{i-1}((1-q)\alpha n_j + \beta t) \\ & + \frac{1}{(1-q)^2 + q(1-\sigma)}[(1-q)\alpha n_i + \beta t] \\ & + Q_{NF}\left[\frac{1}{1-q}\alpha N + \frac{1}{(1-q)^2}\beta t + \frac{1}{1-q\sigma}\sum_{j=i+1}^{K}((1-q)\alpha n_j + \beta t)\right]\Bigg\} \end{aligned} \tag{19}$$

5 Cost optimization

Equation (19) can be rearranged, resulting in the following simplified form:

$$C(G) = A(K) + BH(G) \tag{20}$$

where

$$\begin{aligned} A(K) &= Q_{NF}\left[\alpha N\left(\frac{1}{1-q} + \frac{1-q}{1-q\sigma}\right) + \beta t\left(\frac{1}{(1-q)^2} + \frac{K}{1-q\sigma}\right)\right] \\ B &= \frac{(1-q)}{(1-q\sigma)((1-q)^2 + q(1-\sigma))} \end{aligned}$$

and $H(G)$ is an auxiliary cost measure defined as follows

$$\begin{aligned} H(G) &= q\sum_{i=1}^{K} P(g_i)((1-q)\alpha n_i + \beta t) \\ &+ (1-q)\sum_{i=1}^{K} P(g_i)\sum_{j=1}^{i}((1-q)\alpha n_i + \beta t) \end{aligned} \tag{21}$$

From equation (20), it can be seen that, for a given K, only $H(G)$ actually depends on the grouping chosen for the search sequence G, the rest is independent of it. Thus it will be the case that

$$\min_G C(G) = A(K) + B \cdot \min_G H(G)$$

We can, thus, reduce the problem of minimizing $C(G)$ to the problem of minimizing $H(G)$ for all fixed K, $K = 1, \ldots, N$, and then selecting the K for which a minimal cost is obtained. For the rest of this section we will develop an efficient algorithm to obtain a sequence G that minimizes $H(G)$.

In order to provide an efficient optimization algorithm it is necessary to gain more knowledge about the structure possessed by optimal K–search sequences. In particular it will be our first objective to prove that nodes appear in decreasing order of their probabilities. This is expressed by the following theorem.

Theorem 1 *Let G be an optimal K–search sequence defined over $\mathcal{N}$, then for all $i < K$ and for all a, b such that $a \in g_i$ and $b \in g_{i+1}$, $P_a > P_b$.*

To prove the above theorem we will first show some results about the properties satisfied by optimal search sequences.

Lemma 1 *Let G be an optimal K-search sequence, then, for all $i < K$ the following is true*

$$\frac{P(g_i)}{(1-q)\alpha n_i + \beta t} \geq \frac{P(g_{i+1})}{(1-q)\alpha n_{i+1} + \beta t}$$

Proof: Let $G = (g_1, \ldots, g_K)$ be an optimal K-search sequence for which the above is not true for some i. Let's consider the K-search sequence $G' = (g'_1, \ldots, g'_K)$, where $g'_j = g_j$ for all $j \neq i, j \neq i+1$, and $g'_i = g_{i+1}$, $g'_{i+1} = g_i$. Using cost formula (21) we get,

$$\begin{aligned} C(G') - C(G) &= (1-q)\left[P(g_i)((1-q)\alpha n_{i+1} + \beta t) - P(g_{i+1})((1-q)\alpha n_i + \beta t)\right] \\ &< 0 \end{aligned}$$

Which contradicts the hypothesis of G being optimal. ∎

Thus, once the groups are selected, the order in which they are searched is determined by the quantity $\frac{P(g_i)}{(1-q)\alpha n_i + \beta t}$. The above lemma leads directly to the following corollary,

Corollary 1 *Let $G = (g_1, \ldots, g_K)$ be an optimal K–search sequence, if $n_i > n_{i+1}$ then $P(g_i) > P(g_{i+1})$.*

The above result will be used in the following lemma.

Lemma 2 *Let $G = (g_1, \ldots, g_K)$ be an optimal K–search sequence over $\mathcal{N}$. The quantity $((1-q)\alpha n_{i+1} + \beta t + q\alpha(n_{i+1} - n_i))$ is positive for all $i < K$.*

Proof: If $n_{i+1} \geq n_i$ the above quantity is trivially positive. Thus the only case in which it may be negative is when $n_i > n_{i+1}$. We will use contradiction to prove that this case is not possible. Assume, thus, that $n_i > n_{i+1}$ and the lemma does not hold. We will then have an $i < K$, such that the above quantity is not positive. We will show now that in that case we can construct from G a new K–search sequence with a lower cost than G, thus contradicting the hypothesis about G's optimality.

Let $d \in g_i$ be the element of g_i with the lowest probability, that is, for all $e \in g_i$: $P_e \geq P_d$. Let $G' = (g'_1, \ldots, g'_K)$ be defined such that

$$g'_j = \begin{cases} g_i - \{d\} & j = i \\ g_{i+1} \cup \{d\} & j = i+1 \\ g_j & \text{otherwise} \end{cases}$$

That is, G' is formed from G by moving the lowest probability element of g_i to g_{i+1}. By using the cost formula (21), we can readily verify that,

$$\begin{aligned} H(G') - H(G) &= (1-q)[\\ &\quad P_d((1-q)\alpha n_{i+1} + \beta t + q\alpha(n_{i+1} - n_i)) \\ &+ (1+q)\alpha P_d - (1-q)\alpha P(g_i) \\ &- q\alpha(P(g_i) - P(g_{i+1}))] \end{aligned}$$

From corollary 1 we know that the last term is strictly negative. On the other hand, when $q \leq 1/3$, we would have

$$(1+q)\alpha P_d - (1-q)\alpha P(g_i) \leq \frac{4}{3}\alpha P_d - \frac{2}{3}\alpha n_i P_d \leq \frac{4}{3}\alpha P_d - \frac{2}{3}\alpha 2 P_d = 0$$

Thus, the sum of the last three terms is negative. By hypothesis, the first term is non–positive, thus $H(G') - H(G) < 0$, which is a contradiction of the optimality of G. ∎

We are now finally ready to prove theorem 1.

Proof of Theorem 1. We will prove it by contradiction. Then there is an $i < K$ and elements a, b such that $a \in g_i$ and $b \in g_{i+1}$, and such that $P_a < P_b$. We will now construct a new K–search sequence, G', such that G' has a lower cost than G, thus contradicting the hypothesis about G's optimality.

Let $G' = (g'_1, \ldots, g'_K)$, such that,

$$g'_j = \begin{cases} g_i - \{a\} \cup \{b\} & j = i \\ g_{i+1} \cup \{a\} - \{b\} & j = i+1 \\ g_j & \text{otherwise} \end{cases}$$

That, is, G' is built from G by swapping the positions of a and b. By applying the cost formula (21), we would get,

$$H(G') - H(G) = (P_a - P_b)(1-q)((1-q)\alpha n_{i+1} + \beta t + q\alpha(n_{i+1} - n_i))$$

From lemma 2 we know that the last factor in the above expression is positive. On the other hand $P_a < P_b$ by hypothesis, thus the above expression is negative, contradicting the hypothesis of the optimality of G. ∎

Following theorem 1 we label the set of authoritative nodes such that for $a < N$, $P_a \geq P_{a+1}$. Then, if G is an optimal K–search sequence, we will have

$$g_i = \{a, \ldots, a + n_i - 1\} \qquad \text{where } a = 1 + \sum_{j=1}^{i-1} n_j \tag{22}$$

The problem of determining an optimal G reduces to the problem of determining the optimal values for K and n_i, $i = 1, \ldots, K$.

Let $\mathcal{N}_r = \{N - r + 1, \ldots, N\}$, that is, the set of the last r elements of $\mathcal{N}$. We will denote by $G_{k,r}$ a k–search sequence defined over $\mathcal{N}_r$. We will also denote by P^r the probability distribution P conditioned to the fact that the resource is in $\mathcal{N}_r$. If $G_{k,r} = (g_1, \ldots, g_k)$, we would have that $(g_2, \ldots, g_k)$ would be a $(k-1)$–search sequence over $\mathcal{N}_{r-n_1}$, and we would denote it by $G_{k-1,r-n_1}$. Thus, applying cost formula (21) we can now get,

$$H(G_{k,r}) = [1 - q(1 - P^r(g_1))] \cdot [(1-q)\alpha n_1 + \beta t] \; + \; (1 - P^r(g_1)) \cdot H(G_{k-1,r-n_1}) \tag{23}$$

From (23) it can be seen that if $G_{k,r}$ is optimal, then $G_{k-1,r-n_1}$ also has to be optimal. Thus, an optimal k-search sequence over $\mathcal{N}_r$ is formed by selecting a suitable n_1, and an optimal $(k-1)$–search sequence over $\mathcal{N}_{r-n_1}$. The group formed with the n_1 lowest numbered nodes in $\mathcal{N}_r$ is then inserted in front of G. Thus, once all $k-1$ optimal search sequences have been determined, the only thing left to determine an optimal k sequence is to get the optimal value for n_1. This is expressed in the following, where the asterisk, $(*)$, denotes optimality for given k and r.

$$H(G^*_{k,r}) = \min_{\substack{n_1 \\ 1 \leq n_1 \leq r-k+1}} \{(1 - q(1 - P^r(g_1)))((1-q)\alpha n_1 + \beta t) + (1 - P^r(g_1))H(G^*_{k-1,r-n_1})\} \tag{24}$$

Equation (24) leads directly to a dynamic programming solution for the problem. The initial conditions are given by:

$$\begin{aligned} G^*_{1,r} &= (\mathcal{N}_r) \\ H(G^*_{1,r}) &= (\alpha(1-q)r + \beta t) \end{aligned}$$

Using (24) in conjunction with the above initial conditions, results in the following algorithm;

for $r = 1$ **to** N **do**
    set $G^*_{1,r} = (\mathcal{N}_r)$
    set $H(G_{1,r}) = ((1-q)\alpha r + \beta)$
endfor

for $k = 2$ **to** N **do**
    for $r = k$ **to** N **do**
        find n_1 corresponding to the minimization
            in equation (24).
        set $G^*_{k,r} = (\{N-r+1,\ldots,N-r+n_1\}) \cdot G^*_{r-n1,k-1}$
        set $H(G_{k,r})$ to the minimum value in equation (24)
    endfor
endfor
for $k = 1,\ldots,N$ select $G \doteq G^*_{k,N}$ such that
    $C(G^*_{k,N})$ is minimum.

return G.

6 Numerical Examples

In this section we will present some numerical examples that illustrate the application of the optimization algorithm. We will first need to estimate suitable values for the parameters. To estimate the average length, t, of an interval we decompose it as follows (see figure 2).

t = Average time until start of successful transmission
\+ Message transmission time
\+ Average time until node becomes ready to transmit again

We assume that CSMA/CD is used as the media access protocol and define the message length as our unit of time. Thus, the second term above will have the value 1. Using Lam's delay formula [9], we can determine the value of the first term in t's expansion, which in general will depend on the offered load, λ (in messages per packet transmission time). We assume end–to–end propagation delay of 0.2 in packet transmission time units. For Lam's result to be valid for the above end–to–end propagation delay, $\lambda < 0.4372$[9]. We call this the *limit value* of λ.

Finally we estimate the last term in t's expansion to be equal to Δ, and Δ to be equal to 2 time units. This will allow for the propagation delay of the ML message and the propagation and transmission delays of the RL message, as well as processing time required

for a directory search at a responding node. Assuming the arrival of packets is a Poisson process with rate λ, we can estimate σ (the probability that unrelated transmission occurs during Δ) as $\sigma = 1 - e^{-\lambda\Delta}$.

In all the numerical examples we discuss we consider a system with 100 nodes in the authoritative set. In all cases we assume the value of α to be fixed and equal 1, varying only β. We first study a system in which every node in the authoritative set is equally likely to have the resource, that is $P_a = 1/N$ for all $a \in \mathcal{N}$. Also here, we assume that the protocol operates without message errors, that is $q = 0$. In figure 3 we show the variation with the offered load λ of three costs: the cost of an optimal search sequence, the cost of pure polling and the cost of broadcast for $\beta = 0.4$. In general, as λ grows the average interval length, t, also grows, until it reaches infinity when lambda approaches its limit value. This is apparent in the behavior of the costs shown in the figure. Note that the variation in the cost of polling is steeper than that of the cost of broadcast or the optimal cost. Also observe the convergence of the optimal cost to the cost of broadcast for values of λ approaching its limit value.

In Table 1 we show how the optimal groupings change as β varies. The optimal groupings are represented as sequences of numbers $(a_1, \ldots, a_k)$, where each a_k is the first member of group g_j. Thus $g_j = \{a_j, \ldots, a_{j+1} - 1\}$ for all $j < k$; and $g_k = \{a_k, \ldots, 100\}$. Table 2 shows the change in optimal groupings as λ changes over its range. The changes in optimal grouping do not start being significant until λ starts approaching the limit value. In figure 4 we show the variation of the cost of the optimal k-search sequence as a function of k for $\beta = 0.4, \alpha = 1$ and $\lambda = 0.025$. The optimal grouping is presented in Table 2. It can be observed that the region around the optimal point has a very small slope, suggesting that slight variations from the optimal grouping will not significantly affect the cost (see section 7). In Table 3, we show how a non-zero error probability affects the optimal grouping for particular values of α, β and λ.

Next we study a system with a tri-level probability distribution:

$$P_a = \begin{cases} 0.099641 & a = 1, \ldots, 10 \\ 0.0001 & a = 11, \ldots, 40 \\ 0.00001 & a = 41, \ldots, 100 \end{cases}$$

The variation of the optimal, broadcast and polling costs is similar to that shown in figure 3 for the uniform case, only the cost of polling is lower. In table 4 we show the optimal groupings obtained in an error-free system for different values of β and for $\lambda = 0.4$. It can be seen that broadcast is not favored for a wide range in the value of β. It can also be

observed that the nodes starting the equal probability groups in the tri–level distribution also start groups in the optimal grouping.

7 Assigning Nodes to Multicast Groups

In general, the network will have M nodes and R resources. Each node, x, will have a different authoritative set for a resource, s, which we denote by $\mathcal{N}^{<x,s>}$. It may also have a different probability distribution $P^{<x,s>}$. Thus, for each node and resource, there would be, in general, a different optimal grouping $G^*_{<x,s>}$. The more the number of different groupings used, the larger the number of groups to which a node will have to belong. If we were to implement all the above multicast groupings, each node y, would belong to a potentially large number of groups (up to $M \times R$). Network interfaces typically have a limit on the number of multicast addresses they can recognize efficiently. This means that when the groupings to be used with MLP are assigned, care has to be taken that no node is in more groups than its interface permits.

The problem can be alleviated by grouping different resources into classes. We would then have a different grouping per node and per resource class. This, however, may still not be sufficient to decrease the number of multicast groups needed to a reasonable level.

In general, the most likely situation is that for a given resource class, there is a probability distribution indicating the likelihood that the resource be at a given node in the network. Then, to obtain a good grouping of the nodes, we suggest the following heuristic scheme:

1. Assume that a node "external" to the network is searching for a resource belonging to a particular class, for which a probability distribution is defined over the authoritative set $\mathcal{N}$; where $\mathcal{N}$ is a subset of all the network nodes.

2. Apply the optimization algorithm, thus obtaining the optimal grouping, $G^* = (g_1, \ldots, g_k)$, over $\mathcal{N}$.

3. For a node in the network not in $\mathcal{N}$, the above grouping G^* would also be optimal.

4. Any other node $x \in \mathcal{N}$ will belong to exactly one of the groups $g_i$of G^*. For such a node we use the following search sequence,

$$G_{<x>} = \begin{cases} (g_1, \ldots, g_{i-1}, g_{i+1}, \ldots, g_k) & |g_i| = 1 \\ (g_1, \ldots, g_{i-1}, g_i - \{x\}, g_{i+1}, \ldots, g_k) & |g_i| > 1 \end{cases}$$

It can be easily seen that when these search sequences are used, any particular node will belong to only one multicast group (g_i). In general, this search sequence will be suboptimal. This is due to the fact that an authoritative node will use MLP only when the resource is not found locally. Thus the authoritative set for x would be, $\mathcal{N}^{<x>} = \mathcal{N} - \{x\}$. And the probability distribution over $\mathcal{N}^{<x>}$ would be the same as for $\mathcal{N}$ but conditioned on the fact that x does not have the resource. This, in general, would produce optimal groupings different from the ones given above.

On the average we will thus have a suboptimal multicast group assignment strategy. To investigate the difference in cost from the optimal group assignment, we computed the actual average cost over the whole network incurred when using the heuristic grouping described above for a wide range of cases. We found that the difference from the optimal cost was insignificant. Thus, in a real system, the above approach could be taken, guaranteeing that each node belonged to at most one multicast group per resource class, and thus, the number of multicast addresses required per node will be less than or equal to the number of resource classes.

8 Concluding Remarks

In this paper we have presented a resource location protocol (the MLP) making use of the multicast capabilities of the network. The scheme operates by polling a sequence of multicast groups until the resource is found. This generalizes both pure polling and broadcast search. A system using MLP is modeled and analyzed using a performance measure that takes into account both the cost incurred by the nodes searching their local directories, as well as nodes waiting for their location requests to be satisfied. As a result of the analysis, an expression is derived for the average cost incurred by MLP when a particular sequence of node groups is used. Using this, we develop an algorithm that produces a sequence of node groups minimizing the cost of the protocol. Based on that algorithm, we also show how a near-optimal grouping can be practically set up in a given system.

MLP, as presented, has been assumed to work on a bus network. We believe it can be easily adapted to different topologies, although the cost expression will, in general, be different. We plan to study the adaptation of MLP and the optimization algorithm to other network topologies.

References

[1] A. D. Birrel, R. Levin, R. M. Needham, and M. D. Schroeder, "Grapevine: an exercise in distributed computing," *Communications of the ACM*, vol. 25, pp. 260–274, April 1982.

[2] D. Oppen and Y. Dalal, "The clearinghouse: a decentralized agent for locating named objects in a distributed environment," *ACM Transactions on Office Information Systems*, vol. 1, pp. 230–253, July 1983.

[3] B. Lindsay, "Object naming and catalog management for a distributed database manager," in *Second International Conference on Distributed Computing Systems*, pp. 31–40, IEEE, April, 8–10 1981.

[4] D. Terry, "Caching hints in distributed systems," *IEEE Transactions on Software Engineering*, vol. SE-13, pp. 48–54, January 1987.

[5] M. Ammar, J. Bernabéu–Aubán, and M. Ahamad, "Using Hint Tables to Locate Resources in Distributed Systems," in *INFOCOM'88*, IEEE, March 1988.

[6] P. Dasgupta, R. J. LeBlanc, and E. Spafford, "The clouds project: design and implementation of a fault-tolerant distributed operating system," Tech. Rep. GIT-ICS-85/29, Georgia Institute of Technology, 1985.

[7] M. Ahamad, M. Ammar, J. Bernabéu–Aubán, and Y. Khalidi, "Using Multicast Communication to Locate Resources in a LAN-Based Distributed System," in *Proceedings of the 13th Conference on Local Computer Networks*, IEEE, October 1988.

[8] D. R. Cheriton and T. P. Mann, "A decentralized naming facility," Tech. Rep. STAN–CS–1098, Department of Computer Science, Stanford University, February 1986. Revised version to appear in ACM Transactions on Computer Systems.

[9] S. S. Lam, "A Carrier Sense Multiple Access Protocol for Local Networks," *Computer Networks*, vol. 4, pp. 21–32, 1980.

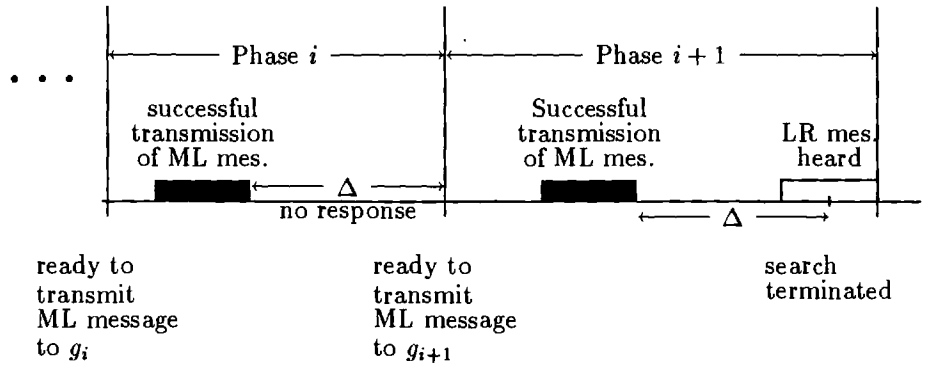


Figure 1: Error–free operation of MLP

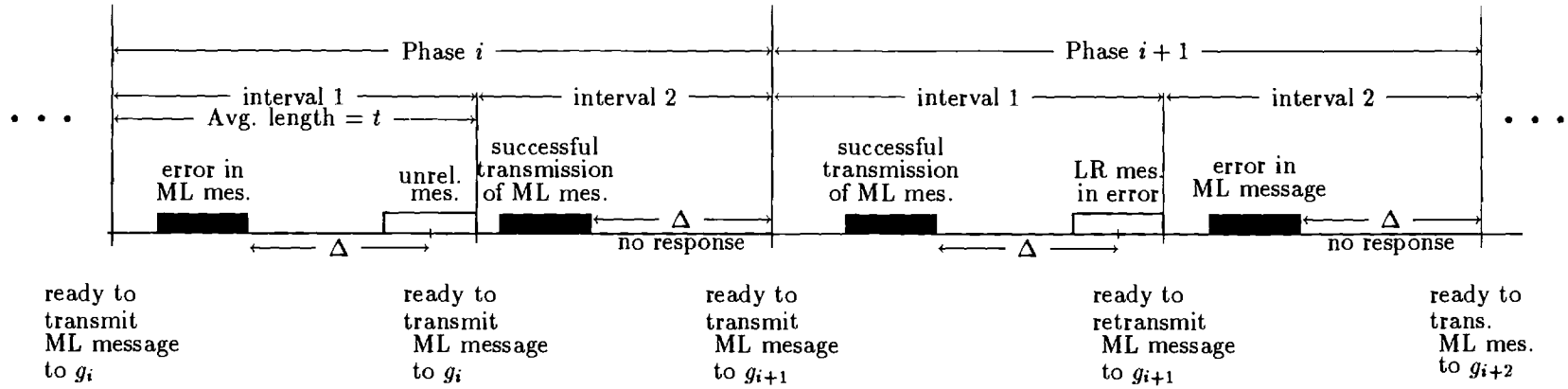


Figure 2: Errors in messages

β	Optimal grouping	C	Poll	Bdcst
0.0	$(1,\ldots,100)$	50.5	50.5	100
0.1	$(1,9,17,24,31,38,44,50,56,61,66,71,$ $75,79,83,86,89,92,94,96,98,99,100)$	55.71	67.85	100.34
0.4	$(1,17,31,44,56,66,75,83,89,94,98,100)$	61.75	119.89	101.37
1.0	$(1,26,47,65,79,90,97)$	69.23	223.98	103.44
4.0	$(1,42,72,92)$	86.29	588.29	110.65
30.0	(1)	203.06	5254.93	203.06

Table1: optimal groupings for uniform distribution, $\lambda = 0.025$ and $\alpha = 1$.

λ	optimal grouping	opt K
0.025000	(1,17,31,44,56,66,75,83,89,94,98,100)	12
0.075000	(1,17,32,45,57,67,76,84,90,95,98,100)	12
0.125000	(1,17,32,45,57,68,77,85,91,96,99)	11
0.175000	(1,18,33,47,59,70,79,87,93,97,100)	11
0.225000	(1,18,34,48,60,71,80,87,93,97,100)	11
0.275000	(1,19,36,51,64,75,84,91,96,99)	10
0.325000	(1,22,40,56,69,80,89,95,99)	9
0.375000	(1,26,48,66,80,91,98)	7
0.400000	(1,32,57,77,91,99)	6
0.425000	(1,33,86)	3

Table 2: Variation of optimal groupings with λ for a uniform system with error-free operation of MLP, where $\beta = 0.4$.

q	Optimal Grouping	C	Poll	Bcast
0.01	$(1,17,32,45,57,67,76,84,90,95,99)$	63.98	123.32	103.47
0.03	$(1,17,32,45,57,68,77,85,92,97)$	65.14	125.12	104.55
0.07	$(1,17,32,46,58,69,79,88,95)$	70.23	132.90	109.12
0.15	$(1,18,33,47,60,72,83,93)$	82.58	151.48	119.55
0.24	$(1,19,36,51,65,78,90)$	100.62	177.77	133.96

Table 3: Optimal grouping for uniform distribution, where $\lambda = 0.025$, $\alpha = 1$ and $\beta = 0.4$.

β	Optimal grouping	C	Poll	Bdcst
0.0	$(1, \ldots, 100)$	5.60	5.60	100
0.1	$(1, 6, 9, 11, 21, 29, 36, 41, 53, 64, 74, 82, 89, 94, 98, 100)$	9.57	13.97	101.50
0.4	$(1, 9, 11, 29, 41, 65, 23, 95)$	15.75	39.09	105.98
1.0	$(1, 11, 41, 76, 96)$	25.16	89.34	114.96
4.5	$(1, 11, 41)$	77.73	382.43	167.30
15.0	$(1, 11, 41)$	160.33	843.00	249.56
600	$(1, 41)$	9016.36	50249.50	9073.64
6600	(1)	98810.04	552688.52	98810.04

Table 4: Optimal grouping for the tri–level distribution, where $\lambda = 0.4$ and $\alpha = 1$.

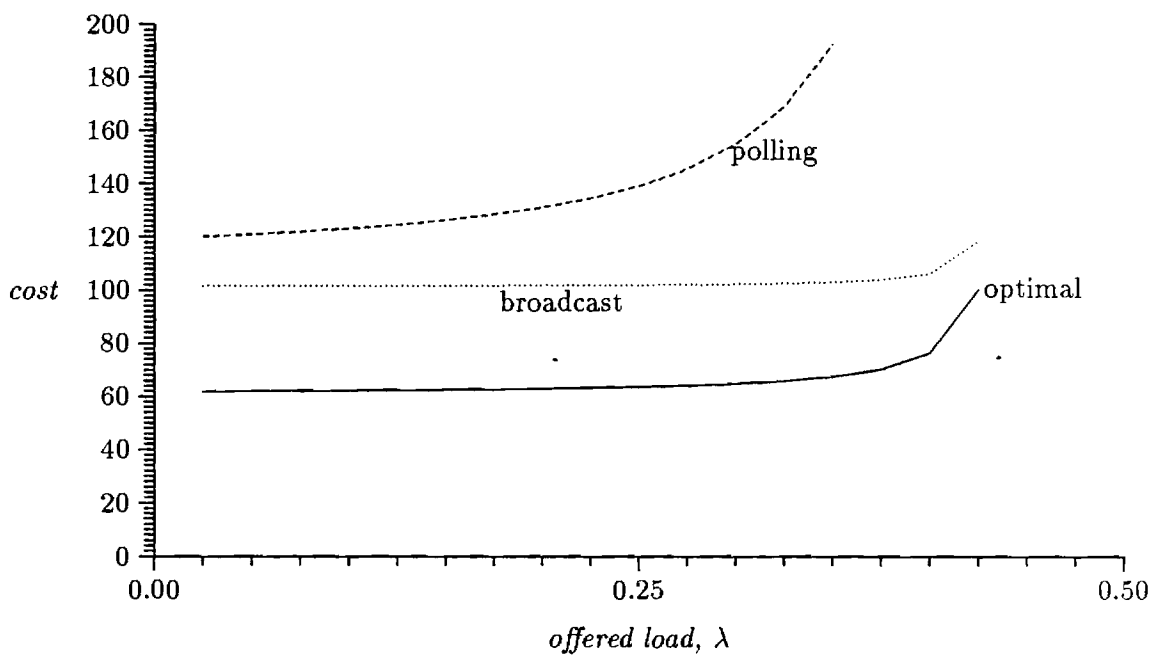


Figure 3: Variation of the optimal,broadcast and polling costs as a function of the offered load for $\beta = 0.4$

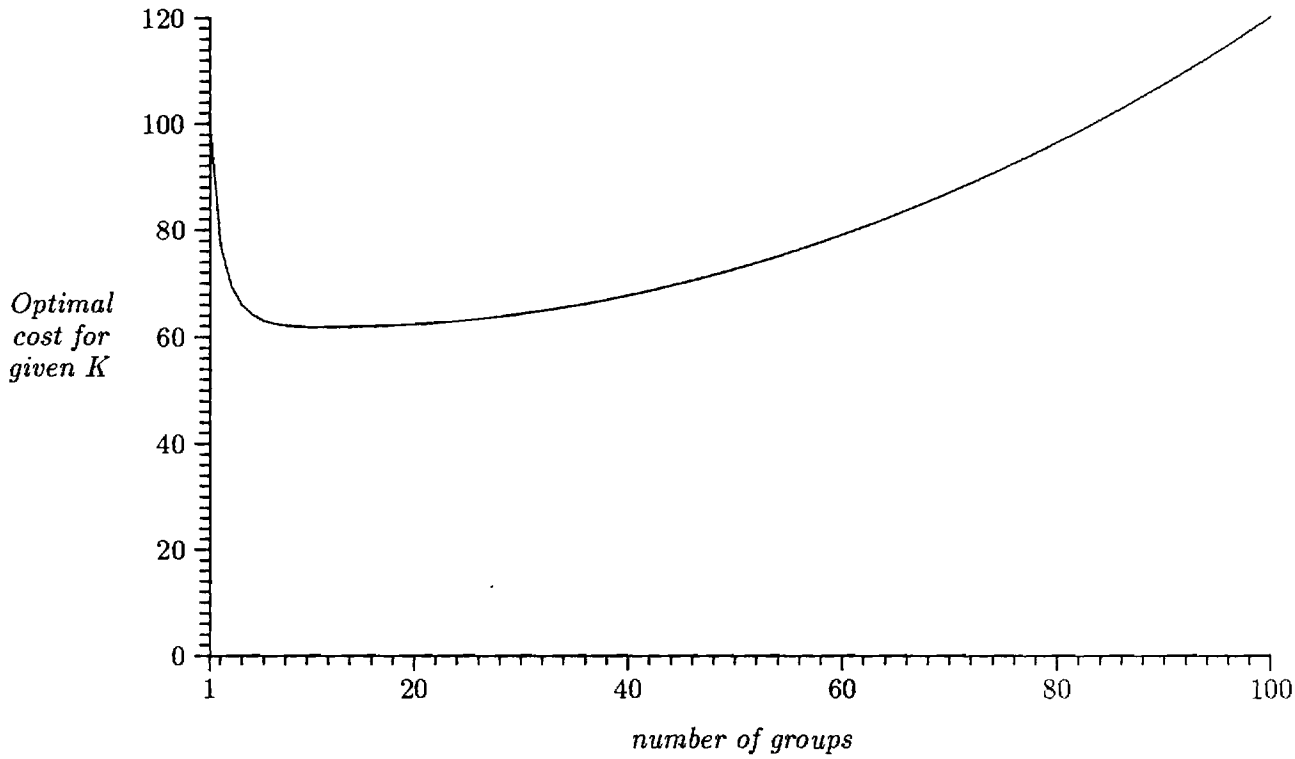


Figure 4: Optimal cost for fixed values of K, when $\lambda = 0.4$, $\alpha = 1$ and $\beta = 0.4$ (the optimal K is 12)

Optimizing Vote and Quorum Assignments for Reading and Writing Replicated Data

Shun Yan Cheung
Mustaque Ahamad
Mostafa H. Ammar

Technical Report GIT-ICS-88/20
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332

August 25, 1988

Abstract

In the weighted voting protocol which is used to maintain the consistency of replicated data, the availability of the data to read and write operations not only depends on the reliabilities of the nodes storing the data but also on the vote and quorum assignments used. We consider the problem of determining the vote assignment and quorum that yields the highest availability in a system where node reliabilities can be different and the mix of the read and write operations is arbitrary. For this purpose, we present an enumeration algorithm that can be used to find the vote and quorum assignments that need to be considered for achieving optimal availability. Also, an analytical method is derived to evaluate the availability of a given system for any vote and quorum assignment. This method and the enumeration algorithms are used to find the optimal vote and quorum assignment for several systems. The enumeration algorithm can also be used to obtain the optimal performance when other measures are considered.

1 Introduction

A distributed system consists of a number of potentially unreliable nodes interconnected via a communication subnetwork. The resources stored at the nodes can be shared and when a node fails, the resources stored at the node become unavailable. Replicating resources at different nodes with independent failure modes can enhance availability and fault tolerance, since a resource could be available even when some nodes have failed. When data is replicated, care must be taken to preserve consistency among the various copies or *replicas*. In addition to increased availability, replication can also provide improved performance of read transactions by reducing the network communication cost since these transactions can access the data from the local replica.

A large number of replica control protocols have been developed to maintain the consistency of replicated data [DGS85]. In this paper, we address the issue of optimizing availability for a voting-based replica control protocol by deriving the optimal settings for the parameters of the protocol. We consider the voting mechanism, because it has proven to be flexible and relatively easy to implement.

Voting has been used for various applications in distributed systems. Its use for synchronizing read and write operations on replicated files was first proposed in [Gif79]. Each file replica is assigned some number of votes and each operation is required to obtain a pre-defined quorum of votes to proceed. To ensure that a read operation returns the value installed by the last write operation, the read and write operations must acquire r and w number of votes respectively such that $r + w > L$, where L is the total number of votes assigned to all copies. The values r and w are called the read and write quorum. Generally, $r + w = L + 1$ is used which ensures that each read quorum has a non-empty intersection with each write quorum. Since all replicas need not be updated when a write operation completes, timestamps or version numbers must be used in order to determine the value that is written most recently. When version numbers are used, each write quorum must also intersect with every other write quorum, i.e., $2w > L$ [Gif79].

A number of replica control protocols have been derived from weighted voting. Eager and Sevcik introduced a dynamic scheme based on voting that allows the system to switch between normal and failure modes [ES83] (which have different values for read and write

quorums). The system can also change the quorum assignment in the schemes presented in [Her87,AT86,JM87] and the vote assignment can be changed in the scheme described in [BGS86]. Other voting based protocols are presented in [Par86,DB85,AA88].

The problem of assigning votes to achieve mutual exclusion is addressed by Garcia-Molina and Barbara in [GB85]. When the quorum for each operation is a majority of all votes assigned, each operation will have mutually exclusive access to the data. In general, mutual exclusion can be guaranteed by defining a set of groups of nodes [Lam78], called a coterie, such that any two groups in a coterie have a non-empty intersection. When voting is used, the groups of nodes that have a majority of the votes constitute a coterie (there exist coteries that cannot be obtained from any vote assignment) [GB85]. In [GB85], it is shown that only a finite set of vote assignments need to be considered to get all coteries that can be obtained from vote assignments. Thus, it is not necessary to deal with the unbounded set of possible vote assignments. In another work, the same authors have considered the problem of selecting the vote assignment that results in the highest system availability for mutual exclusion [BG87]. For a system where all node reliabilities are the same, they derived the optimal vote assignment. is a function of the node reliabilities.

In a related work [AA87], the authors evaluate the use of the voting mechanism to manage read and write transactions. For a systems where node reliabilities are identical, values are derived for the optimal degree of replication and for the optimal read quorum.

In this paper, we study the problem of finding the vote and quorum assignments that result in the best system availability for read and write operations in a system where node reliabilities may be different. For this problem, the proportion of read and write operations and the read and write quorums have also to be taken into account. Although the number of vote assignments is theoretically unbounded, similar to mutual exclusion, the set of vote assignments that should be considered for best performance for reading and writing is also finite. We first present an algorithm that generates these vote assignments and then describe a method to compute the availability of the system with a given vote assignment for reading and writing replicated data. By applying this method to the vote assignments enumerated, the optimal vote and quorum assignment can be determined.

The paper is organized as follows. We describe the enumeration algorithm in Section 2. Section 3 presents a model of the system and the analysis that derives a method for finding

the availability. Some numerical examples are presented in Section 4 and we conclude the paper in Section 5.

2 Enumerating Vote Assignable Read Sets

2.1 Definitions

Let N be the number of nodes that store copies of a data item. We number these nodes from 1 to N. Copy i resides at node i and let $U_N = \{1, 2, \ldots, N\}$ be the universe of the copies. There are two types of operations allowed on the copies, read and write, and each operation must acquire the consensus of a number of copies to proceed. A *read group* is a minimal group of copies such that a read operation can proceed if all copies in the group are available (i.e., the nodes where the copies are stored have not failed). Thus, failure to acquire the consensus of all copies in a read group causes the read operation to block or abort. A *read set* Q_r is a collection of read groups satisfying the following *non-containment* property, which is a result of the fact that each read group is minimal:

$$\forall\, G, H \in Q_r :\ G \not\subset H \text{ and } H \not\subset G$$

A write operation can proceed only if it can acquire the consensus of copies that constitute a write group. The *write set* Q_w corresponding to a given read set Q_r is unique and consists of write groups H that satisfy the non-containment property and also for each $H \in Q_w$:

$$\forall\, G \in Q_r : G \cap H \neq \phi$$

We assume in this paper that timestamping is used to identify the current copy. When version numbers are used for this purpose, the intersection of any two write groups must also be non-empty. The results of this paper can trivially be modified to accommodate this case.

In voting [Gif79], a special subset of read/write sets is used, namely those that can be obtained from some vote assignment and we will call these read/write sets *vote assignable*. A vote assignment is a vector $\underline{V}_N = (v_1, v_2, \ldots, v_N)$ where v_i $(1 \leq i \leq N)$ is a non-negative integer representing the number of votes assigned to copy i. We define $L(\underline{V}_N)$ to be the total number of votes assigned to the copies, or $L(\underline{V}_N) = \sum_{i=1}^{N} v_i$. Let group $G = \{g_1, g_2, \ldots, g_k\}$

be a set of copies where copy g_1 has the least number of votes and we denote by $v(G)$ the sum of the votes assigned to copies in G. The group G is a *tight group* of s votes if and only if the copies in G collectively have at least s votes and removal of any copy (in particular the copy g_1 which has the least number of votes) from G leaves a group with less than s votes. In other words,

$$G \text{ is a tight group of } s \text{ votes} \iff s \leq v(G) \leq (s-1) + v_{g_1}$$

A *read* group of r votes is a tight group of r votes. The value r is called the read quorum. A *read set* of r votes consists of all the read groups of r votes. A vote assignment $\underline{V}_N$ and a read quorum r uniquely identify a read set. However, the same read set can be obtained using different vote assignments and/or different read quorums. For example, the read set $\{\{1,2\},\{3\}\}$ can be obtained from $\underline{V}_1 = (1,1,2)$ and $r = 2$ and $\underline{V}_2 = (2,3,5)$ and $r = 4$. For each read set with a quorum of r votes there exists a write set with a quorum of $L + 1 - r$ votes. The write set for a given read set is unique and we can limit our attention in the enumeration process to only the read sets.

For a system with N nodes, we use $Q_N(r,\underline{V}_N)$ to denote the read set obtained from the vote assignment $\underline{V}_N$ when the read quorum is r. For $1 \leq r \leq L(\underline{V}_N)$ the read set is well-defined. If $r > L(\underline{V}_N)$ then we define $Q_N(r,\underline{V}_N) = \phi$, i.e., no read group can be formed. For $r \leq 0$, we define $Q_N(r,\underline{V}_N)$ to be $\{\phi\}$, i.e., a read operation requires no consensus.

We denote the universe of vote assignable read sets of N copies by Ω_N. Since each member of Ω_N is obtained from a vote assignment $\underline{V}_N$ and a quorum r, Ω_N also defines a set of $(\underline{V}_N,r)$ pairs. Two read sets R and S are *isomorphic* if and only if there is a permutation π of integers $1, \ldots, N$ such that when we replace each i in S by $\pi(i)$, we obtain R. If the vote assignment $\underline{V}_N$ is permuted and the read quorum r is kept at the same value, the resulting read set will be isomorphic to $Q_N(r,\underline{V}_N)$. (Permuting the vote assignment will, in general, affect the performance in a system with different node reliabilities.) A collection of read sets E_N is an *enumeration* [GB85] if every read set in Ω_N is either in E_N or is isomorphic to one in E_N and no two read sets in E are isomorphic. Note that $E_N \subseteq \Omega_N$ and E_N can be obtained from Ω_N by choosing one representative from each isomorphic class. Conversely, Ω_N can be obtained from E_N by applying all possible permutations of $1,2,\ldots,N$ to the members of E_N. For optimization purposes, Ω_N is the space that must be searched to find the best vote assignable read set.

2.2 Generating All Vote Assignable Read Sets

We now present an algorithm that can be used to generate Ω_{N+1} when Ω_N is given. Since $\Omega_1 = \{\{\phi\},\{\{1\}\},\phi\}$, the algorithm can be used, in principle, to find Ω_N, for any value of N. The algorithm is derived from the results of the following lemma that states how the read set of a system with N copies changes when the system is expanded by creating a new replica (copy $N+1$) which is assigned v_{N+1} votes.

Lemma 2.1 *Expanding the Vote Assignment*

Let $\underline{V}_N = (v_1, v_2, \ldots, v_N)$ be a vote assignment to N copies and $\underline{V}_{N+1} = (v_1, v_2, \ldots, v_N, v_{N+1})$ be a vote assignment to $N+1$ copies, such that the first N values are the same as in $\underline{V}_N$. Then,

$$\begin{aligned} Q_{N+1}(r,\underline{V}_{N+1}) \;=\; & Q_N(r,\underline{V}_N) \bigcup \\ & \{G \cup \{N+1\} \mid G \in Q_N(r - v_{N+1},\underline{V}_N) \wedge G \notin Q_N(r,\underline{V}_N)\} \end{aligned}$$

Proof: See Appendix A.

We know from Lemma 2.1 that if there is a vote assignable read set $Q_{N+1}(r,\underline{V}_{N+1})$ of $N+1$ copies then there must exist Q_1 and Q_2 ($Q_1 = Q_N(r,\underline{V}_N)$ and $Q_2 = Q_N(r - v_{N+1},\underline{V}_N)$) such that $Q_{N+1}(r,\underline{V}_{N+1})$ is related to Q_1 and Q_2 as stated in the lemma. The algorithm presented in Figure 1 uses this fact as it generates read sets of a system of $N+1$ copies by combining every pair of read sets of N copies using the relationship defined by Lemma 2.1. Notice that since Q_1 and Q_2 in Lemma 2.1 have the same vote assignment and we are combining all pairs (even those which are obtained from different vote assignments), the algorithm has to check that the resulting set can be obtained by some vote assignment. The output of the algorithm, Ω, will be Ω_{N+1}. This follows from the fact that each read set $Q \in \Omega_{N+1}$ can be written as a combination of some pair of read sets in Ω_N (as given by Lemma 2.1) and every possible pair of read sets of Ω_N is combined by the algorithm in the manner given by Lemma 2.1, hence Q will be generated.

Note that Q, the set generated by the statement (†) of the algorithm, may not satisfy the non-containment property for read sets. For example, $Q_1 = \{\{1\}\}$, $Q_2 = \{\{1,2\}\}$ and

Ω := ϕ; (Ω is the output of the algorithm)
for every $Q_1 \in \Omega_N$ do
    for every $Q_2 \in \Omega_N$ do
        Q := $Q_1 \bigcup \{G \cup \{N+1\} \mid G \in Q_2 \wedge G \notin Q_1\}$; ....(†)
        if Q is obtainable from some $(\underline{V}_{N+1}, r)$ and $Q \notin \Omega$ then
            Record $(\underline{V}_{N+1}, r)$;
            $\Omega := \Omega \cup Q$;

Figure 1: Algorithm 1

$N+1 = 3$, the set $Q = \{\{1\},\{1,2,3\}\}$ is generated which violates the non-containment property. However, such sets are not vote assignable and will not be included in Ω. The statement (†) also generates read sets (satisfying the non-containment property) that are not vote assignable.

Determining whether the set Q is vote assignable can be done by formulating a Linear Program (LP) from the groups of Q. We define the following set of groups of $N+1$ copies:

$$Y(Q) = \{H \in 2^{U_{N+1}} \mid \quad H \text{ is not a superset of any group of } Q \text{ or}$$
$$H \text{ is a proper subset of a group of } Q\}$$

where $2^{U_{N+1}}$ is the set of all subsets of the universe of $N+1$ nodes. When a group G is in Q, any proper subset of G must have less than r votes. Also when a group H does not contain any group of Q, it cannot have r or more votes (if it did then H or its subset must be in Q). The set $Y(Q)$ is thus the set of all groups that do not have the required quorum of r votes. We use this property of $Y(Q)$, coupled with the fact that all the groups in Q must have at least r votes, to formulate the following LP,

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{N+1} v_i + r \\
\text{s.t.:} \quad & \forall\, G \in Q : \sum_{g \in G} v_g \geq r \\
& \forall\, H \in Y(Q) : \sum_{h \in H} v_h \leq r - 1 \\
& v_i \geq 0, i = 1, 2, \ldots, N+1 \\
& r \geq 1
\end{aligned}
$$

If the LP does not have a solution, then there are no values $\underline{V}_{N+1}$ and r that satisfy the constraints. If the LP does produce a solution, the values of $\underline{V}_{N+1}$ and r are rational and since multiplying $\underline{V}_{N+1}$ and r by the same value will not affect the resulting read set, we can always convert the solution to an integer vote assignment and quorum. In principle, we are only concerned with obtaining a feasible solution to the LP. However, the complexity of the performance analysis method described in Section 3 is a function of the votes v_i and quorum r. We are thus using the indicated objective function.

2.3 Generating An Enumeration of Vote Assignable Read Sets

Observe that the complexity of Algorithm 1 depends on the size of the input set Ω_N. Also note that, for a given N, we only need to generate the enumeration set E_N from which Ω_N can be obtained. We next describe an algorithm which will generate the set E_{N+1} given as input the set E_N. Due to the fact that, in general, E_N is much smaller than Ω_N (e.g., $\mid E_5 \mid = 119$ and $\mid \Omega_5 \mid = 3287$), the algorithm will require less CPU time.

An algorithm to generate the set E_{N+1} can be derived from Algorithm 1 by only including one member from a class of isomorphic read sets. The following lemma provides a simple technique to achieve this and the resulting enumeration algorithm is shown in Figure 2.

Lemma 2.2 *Equality of Isomorphic Read Sets of non-decreasing Vote Assignments*

Let $\underline{V}_N = (v_1, v_2, \ldots, v_N)$ and $\underline{W}_N = (w_1, w_2, \ldots, w_N)$ be two non-decreasing vote assignments, i.e., $v_1 \leq v_2 \leq \ldots \leq v_N$ and $w_1 \leq w_2 \leq \ldots \leq w_N$. $Q_N(r, \underline{V}_N)$ and $Q_N(s, \underline{W}_N)$

are isomorphic if and only if $Q_N(r,\underline{V}_N) = Q_N(s,\underline{W}_N)$.

Proof: See Appendix A.

```
E := φ; (E is the output of the algorithm)
for every Q1 ∈ E_N do
    for every Q2 ∈ E_N do
        Q := Q1 ⋃ {G ∪ {N+1} | G ∈ Q2 ∧ G ∉ Q1};
        if Q is obtainable from some (V_{N+1}, r) then
            V'_{N+1} := sort(V_{N+1}); (sort in non-decreasing order)
            if Q_{N+1}(r, V'_{N+1}) ∉ E then
                Record (V'_{N+1}, r);
                E := E ∪ Q_{N+1}(r, V'_{N+1});
```

Figure 2: Algorithm 2

The above lemma shows that in each class of isomorphic read sets, there is *exactly one* member that is obtained from a non-decreasing vote assignment, i.e., two read sets that are obtained from non-decreasing vote assignments are either equal or non-isomorphic.

The following theorem shows that Algorithm 2 will correctly generate E_{N+1} given that E_N is correct.

Theorem 2.1 *Enumeration of Vote Assignable Read Sets*

Let E be the output generated by Algorithm 2, then $E = E_{N+1}$ which is the enumeration of Ω_{N+1} with read sets that have non-decreasing vote assignments.

Proof:
Claim 1: Every vote assignable read set of $N+1$ copies is either in E or is isomorphic to a read set in E.

Let $Q_{N+1}(r,\underline{V}_{N+1})$ be an arbitrary read set of $N+1$ copies and $\underline{V}'_{N+1}$ be the non-decreasing vote assignment obtained from $\underline{V}_{N+1}$. Then $Q_{N+1}(r,\underline{V}'_{N+1})$ and $Q_{N+1}(r,\underline{V}_{N+1})$

are equal or isomorphic and by Lemma 2.1 we can write $Q_{N+1}(r,\underline{V}'_{N+1})$ as:

$$Q_{N+1}(r,\underline{V}'_{N+1}) = Q_N(r,\underline{V}'_N) \bigcup \{G \cup \{N+1\} \mid G \in Q_N(r - v'_{N+1},\underline{V}'_N) \wedge G \notin Q_N(r,\underline{V}'_N)\}$$

and since $Q_N(r,\underline{V}'_N)$ and $Q_N(r - v'_{N+1},\underline{V}'_N)$ are in E_N, $Q_{N+1}(r,\underline{V}'_{N+1})$ is included in E.

Claim 2: There are no isomorphic read sets in E.

Each vote assignment is sorted in non-decreasing order and by Lemma 2.2 we conclude that no two read sets in E are isomorphic.□

Table 1 contains the vote assignments, quorums and read sets of E_4 which is produced by repeating the enumeration algorithm three times starting with $E_1 = \{\{\phi\},\{\{1\}\},\phi\}$. The read sets $\{\phi\}$ and ϕ are not included in the table but are elements of E_4. We have also generated E_5, E_6 and E_7. These enumerations have 119, 1113 and 29375 members respectively. [1]

For a given N, once the set E_N is obtained, optimal vote and quorum assignments can be determined for any particular performance measure by an exhaustive search. In what follows we illustrate this by considering the optimization of the availability of replicated data for read and write operations.

3 Performance Analysis of Weighted Voting

We now develop a method for evaluating the availability of replicated data for a given vote assignment and read quorum when the system consists of N nodes and each node stores a copy of the data item. This method will be used to evaluate the performance of the vote and quorum assignments enumerated in the previous section in order to determine the best one.

A node is prone to failure, and when failed, it cannot participate in any operation. For each node i, $i = 1,\ldots,N$, we assume that the mean time-to-fail is $\frac{1}{\gamma_{Fi}}$ and the mean time-

[1]The enumeration method has so far produced only integral valued vote assignments for every LP generated. It is known that when the constraint matrix of the LP is totally unimodular, the solution is integral valued [VD68]. The constraint matrices generated by the algorithm do not satisfy this property.

to-repair is $\frac{1}{\gamma_{Ri}}$. We also define the parameter p_i which represents the proportion of time the node is operational. We thus have

$$p_i = \frac{\frac{1}{\gamma_{Fi}}}{\frac{1}{\gamma_{Fi}} + \frac{1}{\gamma_{Ri}}} = \frac{\gamma_{Ri}}{\gamma_{Fi} + \gamma_{Ri}} \tag{1}$$

The parameter p_i will be called the *reliability* of node i and can be viewed as the steady state probability that the node is operational. We will use the *reliability vector* $\underline{P}$ to denote $(p_1, \ldots, p_N)$. The vote assignment under consideration is denoted by $\underline{V} = (v_1, \ldots, v_N)$ and we define $L = \sum_{i=1}^{N} v_i$ (we drop the subscript N from our earlier notation for simplicity).

The availability for operations requiring s votes is the proportion of time (or steady state probability) that the total number of votes from all operational nodes is equal to or exceeds s. We denote this probability for given quorum s, reliability vector $\underline{P}$ and vote assignment $\underline{V}$ by $\alpha_s(\underline{P}, \underline{V})$ ($s = r$ for read access, $s = w$ for write access and $r + w = L + 1$). We use the system availability α as the performance measure in the analysis. The system availability for read/write transactions is equal to $\alpha(\underline{P}, \underline{V}) = f\alpha_r(\underline{P}, \underline{V}) + (1 - f)\alpha_w(\underline{P}, \underline{V})$, with f being the fraction of read operations.

In order to derive an expression for $\alpha_s(\underline{P}, \underline{V})$ for given values of s, $\underline{P}$ and $\underline{V}$, we define the state of the system $\underline{n} = (n_1, \ldots, n_N)$ where $n_i = 1$ if node i is operational and $n_i = 0$ otherwise. Let $P(\underline{n})$ be the steady state probability that the system is in state $\underline{n}$. Observe that $P(\underline{n})$ is the probability that all nodes with $n_i = 1$ are operational and that all nodes with $n_i = 0$ have failed. Thus we have

$$P(\underline{n}) = \prod_{i=1}^{N} (1 - p_i)^{(1-n_i)} p_i^{n_i} = \left[\prod_{i=1}^{N} (1 - p_i)\right] \prod_{i=1}^{N} \left(\frac{p_i}{1 - p_i}\right)^{n_i} \tag{2}$$

We next obtain an expression for $Q(m)$, the steady state probability that the total number of votes available in the system is m. This is given by the sum of all state probabilities in equation (2) such that $C(\underline{n}, \underline{V}) = \sum_{i=1}^{N} n_i v_i = m$. Thus we get for $m = 0, 1, 2, \ldots, L$

$$Q(m) = \sum_{all\ \underline{n}\ s.t.\ C(\underline{n},\underline{V})=m} P(\underline{n}) = \frac{1}{G} h(\underline{V}, m) \tag{3}$$

where we define

$$\frac{1}{G} = \prod_{i=1}^{N} (1 - p_i) \tag{4}$$

$$h(\underline{V},m) = \sum_{all\ \underline{n}\ s.t.\ C(\underline{n},\underline{V})=m} \prod_{i=1}^{N} q_i^{n_i} \tag{5}$$

and $q_i = \frac{p_i}{1-p_i}$. Since the operation requires s votes, we obtain that

$$\alpha_s(\underline{P},\underline{V}) = \sum_{m=t}^{L} Q(m) = \frac{1}{G} \sum_{m=t}^{L} h(\underline{V},m) \tag{6}$$

Computing $h(\underline{V},m)$ can be accomplished by observing that [2]

$$\begin{aligned} h(\underline{V},m) &= \sum_{all\ \underline{n}\ s.t.\ C(\underline{n},\underline{V})=m,n_N=0} \prod_{i=1}^{N} q_i^{n_i} + \sum_{all\ \underline{n}\ s.t.\ C(\underline{n},\underline{V})=m,n_N=1} \prod_{i=1}^{N} q_i^{n_i} \\ &= \sum_{all\ \underline{n}\ s.t.\ C(\underline{n},\underline{V})=m,n_N=0} \prod_{i=1}^{N} q_i^{n_i} + q_N \sum_{all\ \underline{n}\ s.t.\ C(\underline{n},\underline{V})=m-v_N,n_N=0} \prod_{i=1}^{N} q_i^{n_i} \end{aligned}$$

Alternatively we can write

$$h(\underline{V},m) = h^{-N}(\underline{V},m) + q_N h^{-N}(\underline{V},m-v_N) \tag{7}$$

where $h^{-N}(\underline{V},m)$ is the same as $h(\underline{V},m)$ except that it is evaluated for a system with node N removed, i.e.,

$$h^{-N}(\underline{V},m) = \sum_{all\ \underline{n}\ s.t.\ C^{-N}(\underline{n},\underline{V})=m} \prod_{i=1}^{N-1} q_i^{n_i}$$

where $C^{-N}(\underline{n},\underline{V}) = \sum_{i=1}^{N-1} n_i v_i$.

For a given $\underline{V}$, the algorithm given in Figure 3 that is derived from (7) can be used to evaluate $H(m) = h(\underline{V},m)$.

If the vote assignment and read quorum of each read set in Ω_N is given, the algorithm described in Figure 3 can be used to compute $\alpha(\underline{P},\underline{V})$ for each ($\underline{V}$,r) corresponding to read sets in Ω_N. The ($\underline{V}$,r) that yields the highest value for $\alpha(\underline{P},\underline{V})$ is the optimal vote and quorum assignment. However, if the nodes are labeled such that $p_1 \leq p_2 \leq \ldots \leq p_N$, we need only consider the non-decreasing vote assignments which correspond to the members of E_N generated by Algorithm 2. In other words, the members in $\Omega_N - E_N$ need not be

[2] This is the same technique used to define the basic relationship for the convolution algorithm used to evaluate closed queueing networks [Buz73].

```
Initialize:  H(0) := 1; H(1), H(2), ..., H(L) := 0;
for i := 1 to N do
    for m := L downto v_i do
        H(m) := H(m) + q_i * H(m - v_i)
    end;
end;
```

Figure 3: Algorithm for Computing $h(\underline{V}, m)$.

considered for optimizing availability since they are clearly suboptimal. [3] This follows from the intuitive idea that the best vote assignment is the one that assigns more votes to nodes with higher reliability which is stated as the following theorem.

Theorem 3.1

Given a reliability vector $\underline{P}$ and a vote assignment $\underline{V}$, where, without loss of generality, $p_1 \leq p_2 \leq \ldots \leq p_N$, $v_1 \leq v_2 \leq \ldots \leq v_N$. Then,

$$\alpha(\underline{P}, \underline{V}) \geq \alpha(\underline{P}, \underline{V}')$$

for any permutation $\underline{V}'$ of the vote assignment $\underline{V}$.

Proof: See Appendix B.

4 Numerical Examples

We demonstrate the use of the results derived in the previous sections by analyzing systems of 5 and 7 nodes. For these systems, we show the optimal vote and quorum assignment and the resulting system availability.

Table 2 shows the optimal vote and quorum assignments for two systems of 5 nodes for

[3] Note, however, that for a different performance measure, all elements of Ω_N may have to be considered.

various values of the transaction mix f. Since the system considered in the first column is relatively homogeneous (four nodes have the same reliability which is 0.8 and the reliability of the other node is 0.9), either the uniform vote assignment is optimal (each copy gets one vote) or the node with higher reliability is assigned an extra vote. This does not hold, as shown in column two of the table, when the reliability of two nodes is 0.9. In this case, the optimal vote assignment is not uniform for all cases except when f is close to 0 or 1.

We show the system availability in Figure 4 for the vote and quorum assignments of Table 2 when the reliability of 3 nodes is 0.8 and it is 0.9 for the other two nodes. Clearly, no single vote and quorum assignment can provide optimal availability for all values of f. For example (2,2,2,3,3) with $r = 6$ is optimal for $f = 0.6$ but when f changes to 0.8, the optimal assignment becomes (1,1,1,2,2) with $r = 3$. Also, note that the availability is not very sensitive to a change in f for some of the vote and quorum assignments.

The optimal system availability as a function of f for the two systems considered above and one where the reliability of each node is 0.8, is shown in Figure 5. The plot for the system of Figure 5 is obtained from the plots of Figure 4 by taking the highest system availability for a given f. Although the optimal availabilities of these systems are similar when f is close to 0 or 1, for other values of f, α increases by almost 2% when the reliability of one node changes from 0.8 to 0.9. Also, the optimal availability for each of the systems is not sensitive to a change in f when f is not close to 0 or 1.

To further illustrate how the optimal vote assignment depends on the node reliabilities and f, in Table 3 we also consider a system of 7 nodes, where the reliability of each node is different ($p_1 = 0.65$, $p_2 = 0.7$, $p_3 = 0.75$, $p_4 = 0.8$, $p_5 = 0.85$, $p_6 = 0.9$ and $p_7 = 0.95$). For $f = 0.9$, we see that the most reliable node is assigned 7 votes while the least reliable node gets only one vote. For $f = 0.8$, the difference in the votes is even more marked.

Finally, to show that the availability obtained from optimal vote and quorum assignment can be appreciably higher than that of the commonly used quorums with a uniform vote assignment (i.e., when $v_i = 1$ for all i), we consider the data in Table 4. In the system, two nodes are highly reliable ($p = 0.9$) while the others only have a reliability of 0.6. We see that when $f = 0.8$, the optimal availability is about 9% higher compared to the read majority/write majority quorum. Even when the optimal quorum, r_{opt}, is considered for a uniform vote assignment in this system, the availability is still 5.5% lower than that

achievable with optimal vote and quorum assignment. The read one/write all quorum has poorer availability for all values of f except when f is 0.999. Thus, compared to the commonly used quorums with each node having a single vote, the optimal vote and quorum assignment provides better system availability.

5 Concluding Remarks

We have presented a method for obtaining an enumeration of vote assignable read sets and their corresponding vote assignments and quorums. We have also developed an efficient method for finding the availability for any vote assignment when the reliability of the nodes and the read/write transaction mix are given. The use of this method was demonstrated by finding the vote assignment and quorum that yields the highest system availability in systems with different node reliabilities. It should be emphasized that the enumeration of the vote assignable read sets can also be used in the optimization of other performance measures. In future research, we plan to study the effect of the communication network and the performance of dynamic voting schemes. We will also develop heuristics for obtaining near optimal vote assignments for larger systems and evaluate them against the optimal assignment obtained by our work.

Appendix A

This appendix describes the proofs of the lemmas used in Section 2.

Lemma A.1 *Removing a Copy from a Read Group*

Let $\underline{V}_N = (v_1, v_2, \ldots, v_N)$ be a vote assignment to N copies. Let $G = \{g_1, g_2, \ldots, g_k\}$ be a read group in $Q_N(r, \underline{V}_N)$. Then, $\forall g_i \in G : G - \{g_i\} \in Q_N(r - v_{g_i}, \underline{V}_N)$.

Proof:

Since $v(G) \geq r$, $v(G - \{g_i\}) \geq r - v_{g_i}$ for any $g_i \in G$. The only reason that $G - \{g_i\} \notin Q_N(r - v_{g_i}, \underline{V}_N)$ is when it is not tight, i.e., there is a copy $g_k \in G$ such that $v(G - \{g_i\}) \geq r - v_{g_i} + v_{g_k}$. However, this implies that $v(G) \geq r + v_{g_k}$ which is a contradiction since G is tight.

Lemma A.2 *Adding a Copy to a Read Group*

Let $\underline{V}_N = (v_1, v_2, \ldots, v_N)$, $\underline{V}_{N+1} = (v_1, v_2, \ldots, v_{N+1})$ and let $G \in Q_N(r, \underline{V}_N)$. Then,
If $v(G) \geq r + v_{N+1}$, then $G \in Q_{N+1}(r + v_{N+1}, \underline{V}_{N+1})$,
otherwise, $G \cup \{N+1\} \in Q_{N+1}(r + v_{N+1}, \underline{V}_{N+1})$.

Proof:

If $v(G) \geq r + v_{N+1}$, then G is also a tight group of $r + v_{N+1}$ votes, or $G \in Q_{N+1}(r + v_{N+1}, \underline{V}_{N+1})$.

If $v(G) < r + v_{N+1}$, then G does not have a sufficient number of votes to be a group of $Q_{N+1}(r + v_{N+1}, \underline{V}_{N+1})$. The group $G \cup \{N+1\}$ will have at least $r + v_{N+1}$ votes and to show that it is tight, let g_1 be the copy in G with the least number of votes. If $v_{N+1} \leq v_{g_1}$, then $N+1$ is the copy with the least number of votes in $G \cup \{N+1\}$ and removal of $N+1$ from $G \cup \{N+1\}$ leaves a group of less than $r + v_{N+1}$ votes and thus G is tight. If $v_{N+1} > v_{g_1}$, then $G \cup \{N+1\}$ satisfies

$$r + v_{N+1} \leq v(G \cup \{N+1\}) \leq (r + v_{N+1} - 1) + v_{g_1}$$

because G satisfies $r \leq v(G) \leq (r-1) + v_{g_1}$. Therefore, $G \cup \{N+1\}$ is tight and $G \cup \{N+1\} \in Q_{N+1}(r + v_{N+1}, \underline{V}_{N+1})$. □

Lemma 2.1 *Expanding the Vote Assignment*

Let vote assignments $\underline{V}_N$ and $\underline{V}_{N+1}$ be as in Lemma A.2. Then,

$$Q_{N+1}(r,\underline{V}_{N+1}) = Q_N(r,\underline{V}_N) \bigcup \{G \cup \{N+1\} \mid G \in Q_N(r - v_{N+1},\underline{V}_N) \wedge G \notin Q_N(r,\underline{V}_N)\}$$

Proof: (by showing LHS $\subseteq$ RHS and RHS $\subseteq$ LHS)

We first show that LHS $\subseteq$ RHS. Let $H \in Q_{N+1}(r,\underline{V}_{N+1})$. If $N+1 \notin H$, then $H \in Q_N(r,\underline{V}_N)$. When $N+1 \in H$, then $H \notin Q_N(r,\underline{V}_N)$. Also, $H - \{N+1\} \notin Q_N(r,\underline{V}_N)$ because H is a tight group of r votes and hence $v(H - \{N+1\}) < r$. Define the group $G_1 = H - \{N+1\}$. We have, by Lemma A.1, that $G_1 \in Q_{N+1}(r - v_{N+1},\underline{V}_{N+1})$, and since $N+1 \notin G_1$, $G_1 \in Q_N(r - v_{N+1},\underline{V}_N)$. Hence, $H \in \{G_1 \cup \{N+1\} \mid G_1 \in Q_N(r - v_{N+1},\underline{V}_N) \wedge G_1 \notin Q_N(r,\underline{V}_N)\}$

We can show that RHS $\subseteq$ LHS by showing that each of the sets in the RHS are subsets of the LHS. When $H \in Q_N(r,\underline{V}_N)$, it must follow that $H \in Q_{N+1}(r,\underline{V}_{N+1})$. Now let $H = G_1 \cup \{N+1\}$, where $G_1 \in Q_N(r - v_{N+1},\underline{V}_N)$ and $G_1 \notin Q_N(r,\underline{V}_N)$. Then, by Lemma A.2, either $G_1 \in Q_{N+1}(r,\underline{V}_{N+1})$ or $G_1 \cup \{N+1\} \in Q_{N+1}(r,\underline{V}_{N+1})$. The former case is not possible because $G_1 \notin Q_N(r,\underline{V}_N)$ and $N+1 \notin G_1$, then $G_1 \notin Q_{N+1}(r,\underline{V}_{N+1})$. Thus, $H = G_1 \cup \{N+1\} \in Q_{N+1}(r,\underline{V}_{N+1})$.□

When two read sets $Q_N(r,\underline{V}_N)$ and $Q_N(s,\underline{W}_N)$ are isomorphic, a permutation π maps $Q_N(r,\underline{V}_N)$ to $Q_N(s,\underline{W}_N)$. The copy $\pi(i)$ in $Q_N(s,\underline{W}_N)$ thus plays the role of the copy i in $Q_N(r,\underline{V}_N)$. The role of a copy is determined by its votes and it follows that π is dependent on the assignments $\underline{V}_N$ and $\underline{W}_N$. The following two lemmas show that when $\underline{V}_N$ and $\underline{W}_N$ are non-decreasing, the read sets $Q_N(r,\underline{V}_N)$ and $Q_N(s,\underline{W}_N)$ are isomorphic if and only if they are equal. In Lemma A.3, we consider the case when two read sets are mapped to each other by a transposition (a transposition is a permutation that exchanges two elements) and in Lemma 2.2 we consider the general case.

Lemma A.3 *Equality of Isomorphic Read Sets under a Transposition*

Let $\underline{V}_N$ and $\underline{W}_N$ be two vote assignments such that for some $a < b$, $v_a \leq v_b$ and $w_a \leq w_b$

and let τ be the transposition $(a\ b)$. Then, for some r and s, $Q_N(r,\underline{V}_N)$ is isomorphic to $Q_N(s,\underline{W}_N)$ under τ if and only if $Q_N(r,\underline{V}_N) = Q_N(s,\underline{W}_N)$.

Proof: By contradiction.

Assume that $Q_N(r,\underline{V}_N)$ and $Q_N(s,\underline{W}_N)$ are isomorphic under τ but not equal. It cannot be the case that $Q_N(r,\underline{V}_N) \subset Q_N(s,\underline{W}_N)$ or $Q_N(s,\underline{W}_N) \subset Q_N(r,\underline{V}_N)$ because isomorphic sets have equal number of groups. Therefore, there is a group $G \in Q_N(r,\underline{V}_N)$ such that $G \notin Q_N(s,\underline{W}_N)$. We split the proof into cases depending on whether a and/or b is an element of G. Write G as $G_1 \cup H$ where G_1 does not contain a and b and $H \subseteq \{a,b\}$.

H cannot be ϕ or $\{a,b\}$, since if $G = G_1$ or $G = G_1 \cup \{a,b\}$ then $G \in Q_N(s,\underline{W}_N)$ contradicting the assumption that $G \notin Q_N(s,\underline{W}_N)$.

Case 1: $G = G_1 \cup \{a\}$.

If $G_1 \cup \{b\} \in Q_N(r,\underline{V}_N)$ also, then $G_1 \cup \{a\} \in Q_N(s,\underline{W}_N)$, contradicting that $G \notin Q_N(s,\underline{W}_N)$. So it must be that $G_1 \cup \{b\} \notin Q_N(r,\underline{V}_N)$ and the following chains of implications can be made from the fact that $G_1 \cup \{a\} \in Q_N(r,\underline{V}_N)$ and $G_1 \cup \{b\} \notin Q_N(r,\underline{V}_N)$:

$$
\begin{aligned}
& G_1 \cup \{a\} \in Q_N(r,\underline{V}_N) \wedge G_1 \cup \{b\} \notin Q_N(r,\underline{V}_N) \\
\Longrightarrow\ & G_1 \cup \{b\} \in Q_N(s,\underline{W}_N) \wedge G_1 \cup \{a\} \notin Q_N(s,\underline{W}_N) \\
\Longrightarrow\ & \forall g \in G_1 : v(G_1) + w_b - w_g < s \wedge v(G_1) < s && [G_1 \cup \{b\} \text{ is tight}] \\
\Longrightarrow\ & \forall g \in G_1 : v(G_1) + w_a - w_g < s \wedge v(G_1) < s && [w_a \leq w_b] \\
\Longrightarrow\ & \text{if } v(G_1 \cup \{a\}) \geq s, \text{ then } G_1 \cup \{a\} \in Q_N(s,\underline{W}_N) \\
\Longrightarrow\ & v(G_1 \cup \{a\}) < s && [G_1 \cup \{a\} \notin Q_N(s,\underline{W}_N)]
\end{aligned}
$$

And,

$$
\begin{aligned}
& G_1 \cup \{a\} \in Q_N(r,\underline{V}_N) \wedge G_1 \cup \{b\} \notin Q_N(r,\underline{V}_N) \\
\Longrightarrow\ & v(G_1 \cup \{b\}) \geq v(G_1 \cup \{a\}) \geq r && [v_b \geq v_a] \\
\Longrightarrow\ & sub(G_1 \cup \{b\}) \in Q_N(r,\underline{V}_N) && [G_1 \cup \{b\} \notin Q_N(r,\underline{V}_N)] \\
\Longrightarrow\ & sub(G_1) \cup \{b\} \in Q_N(r,\underline{V}_N) && [G_1 \cup \{a\} \in Q_N(r,\underline{V}_N)] \\
\Longrightarrow\ & sub(G_1) \cup \{a\} \in Q_N(s,\underline{W}_N) \\
\Longrightarrow\ & v(sub(G_1) \cup \{a\}) \geq s \\
\Longrightarrow\ & v(G_1 \cup \{a\}) > s
\end{aligned}
$$

contradicting the previous conclusion.

The case for $G = G_1 \cup \{b\}$ is similar to the previous one. If $G_1 \cup \{a\} \in Q_N(r,\underline{V}_N)$ also, then $G_1 \cup \{b\} \in Q_N(s,\underline{W}_N)$ which contradicts that $G \notin Q_N(s,\underline{W}_N)$. So it must be that $G_1 \cup \{a\} \notin Q_N(r,\underline{V}_N)$ and as in the previous case, we can derive contradicting conclusions that $v(G_1 \cup \{a\}) < r$ and $v(G_1 \cup \{a\}) > r$.

Lemma 2.2 *Equality of Isomorphic Read Sets of Non-decreasing Vote Assignments*

Let $\underline{V}_N = (v_1, v_2, \ldots, v_N)$ and $\underline{W}_N = (w_1, w_2, \ldots, w_N)$ be two non-decreasing vote assignments. $Q_N(r,\underline{V}_N)$ and $Q_N(s,\underline{W}_N)$ are isomorphic if and only if $Q_N(r,\underline{V}_N) = Q_N(s,\underline{W}_N)$.

Proof: We will use induction to prove that if $Q_N(r,\underline{V}_N)$ and $Q_N(s,\underline{W}_N)$ are isomorphic then they are equal. The converse is obvious.

Let π be a permutation that maps $Q_N(r,\underline{V}_N)$ to $Q_N(s,\underline{W}_N)$, i.e., $Q_N(s,\underline{W}_N) = \pi(Q_N(r,\underline{V}_N))$ where $\pi(Q_N(r,\underline{V}_N))$ denotes the read set obtained by replacing i by $\pi(i)$ in the read set $Q_N(r,\underline{V}_N)$. From the theory of permutations (see for example [Rot84]), we know that a permutation can be factorized into disjoint cycles and this factorization is unique except for the order in which the cycles are written. Also, each r-cycle $(i_1\ i_2\ \ldots i_r)$ can be written as a product of $r-1$ transpositions as follows: $(i_r\ i_1)(i_{r-1}\ i_1)\ldots(i_2\ i_1)$. Therefore, if a permutation π can be factorized into s cycles and cycle i is of length r_i, then π can be factorized into $\sum_{i=1}^{s}(r_i - 1)$ transpositions. The proof of the lemma is by induction on the number of transpositions that constitute the factorization of π. For clarity, we use τ to denote a transposition.

Basis: $\pi = \tau_1 = (a\ b)$

Without loss of generality, assume $a < b$. Then,

$$Q_N(s,\underline{W}_N) = \tau_1(Q_N(r,\underline{V}_N))$$

We thus have by Lemma A.3 that $Q_N(s,\underline{W}_N) = Q_N(r,\underline{V}_N)$ and the basis is proved.

Induction Hypothesis:

Given that $\underline{V}_N$ and $\underline{W}_N$ are non-decreasing, if $Q_N(r,\underline{V}_N)$ and $Q_N(s,\underline{W}_N)$ are isomorphic under a permutation π_k that has a factorization of k or less transpositions, then

$Q_N(r,\underline{V}_N)$ and $Q_N(s,\underline{W}_N)$ are equal.

Induction Step:

We need to show that when $\underline{V}_N$ and $\underline{W}_N$ are non-deceasing and $Q_N(r,\underline{V}_N)$ and $Q_N(s,\underline{W}_N)$ are isomorphic under a permutation π_{k+1} that can be factorized into $k+1$ transpositions, then $Q_N(r,\underline{V}_N) = Q_N(s,\underline{W}_N)$.

Let a be the smallest index such that $\pi_{k+1}(i) = i$; $i < a$ and $\pi_{k+1}(a) \neq a$. We can write π_{k+1} as $\tau_{k+1}\tau_k \ldots \tau_1$ such that $\tau_{k+1} = (a\ b)$ for some $b > a$ and τ_j does not move a for $j = 1,2,\ldots,k$. This can be done by factorizing π_{k+1} into disjoint cycles and re-ordering the cycles such that the cycle containing a is the left-most cycle. After rotating the left-most cycle a number of times such that the last element in the cycle is a, the formula $(i_1\ i_2 \ldots i_r) = (i_r\ i_1)(i_{r-1}\ i_1)\ldots(i_2\ i_1)$ can be used to factorize each cycle to obtain the desired τ_j, $j = 1,2,\ldots,N+1$.

Denote $\tau_k\tau_{k-1}\ldots\tau_1$ by π_k. The permutation π_{k+1} can thus be written as a product of the transposition $\tau_{k+1} = (a\ b)$ and the permutation π_k that has a factorization of k transpositions. Note that the transpositions τ_j, $j = 1,2,\ldots,k$ do not move the copies $1,2,\ldots,a$ and hence $\pi_k(i) = i$ for $i = 1,2,\ldots,a$. We can write

$$\begin{aligned} Q_N(s,\underline{W}_N) &= \pi_{k+1}(Q_N(r,\underline{V}_N)) \\ &= \tau_{k+1}.\pi_k(Q_N(r,\underline{V}_N)) \\ &= \tau_{k+1}(Q_N(r,\underline{U}_N)) \end{aligned}$$

where $\underline{U}_N$ is a vote assignment such that $u_i = v_{\pi_k(i)}$. Since π_k does not move $1,2,\ldots,a$, $u_a = v_a$ and since for all $i > a$, $v_i \geq v_a$, we have $u_b \geq u_a$. By Lemma A.3, we conclude that $Q_N(s,\underline{W}_N) = Q_N(r,\underline{U}_N) = \pi_k(Q_N(r,\underline{V}_N))$, in other words, $Q_N(r,\underline{V}_N)$ and $Q_N(s,\underline{W}_N)$ are isomorphic under a permutation that can be factorized into k transpositions. By the induction hypothesis, we can conclude that $Q_N(s,\underline{W}_N) = Q_N(r,\underline{V}_N)$.□

Appendix B

Theorem 3.1

Given a reliability vector $\underline{P}$ and a vote assignment $\underline{V}$, where, without loss of generality,

$p_1 \leq p_2 \leq \ldots \leq p_N$, $v_1 \leq v_2 \leq \ldots \leq v_N$. Then,

$$\alpha(\underline{P}, \underline{V}) \geq \alpha(\underline{P}, \underline{V}')$$

for any permutation $\underline{V}'$ of the vote assignment $\underline{V}$.

Proof:

We prove the theorem by showing that a vote assignment can be improved by exchanging the votes between two nodes if one of them has higher reliability but is assigned less votes, that is:

Given reliability vector $\underline{P}$ where, without loss of generality, $p_1 \leq p_2 \leq \ldots \leq p_N$. For the two vote assignments

$$\underline{V} = (v_1, v_2, \ldots, v_N)$$

and

$$\underline{V}' = (v_1', v_2', \ldots, v_N')$$

such that for some value l and k:

$$1 \leq l < k \leq N$$
$$v_i' = v_i \text{ for } i \neq k, l;$$
$$v_k' = v_l; \quad v_l' = v_k \text{ and } v_l \leq v_k.$$

we have

$$\alpha_s(\underline{P}, \underline{V}) \geq \alpha_s(\underline{P}, \underline{V}')$$

where s is an arbitrary threshold and $\alpha_s(\underline{P}, \underline{V})$ is given by Equation (6).

From Equation (6) we have

$$\alpha_s(\underline{P}, \underline{V}) = \frac{1}{G} \sum_{m=s}^{L} \sum_{all\ \underline{n}\ s.t.\ C(\underline{n},\underline{V})=m} \prod_{i=1}^{N} q_i^{n_i} \tag{8}$$

We define

$$h^{-lk}(\underline{V}, m) = \sum_{all\ \underline{n}\ s.t.\ C^{-lk}(\underline{n},\underline{V})=m} \prod_{i=1, i\neq l,k}^{N} q_i^{n_i} \tag{9}$$

where $C^{-lk}(\underline{n}, \underline{V}) = \sum_{i=1, i\neq l,k}^{N} n_i v_i$. We thus have that

$$\begin{aligned} G\alpha_s(\underline{P}, \underline{V}) \quad = \quad & \sum_{m=s}^{L} \Big[h^{-lk}(\underline{V}, m) + q_l h^{-lk}(\underline{V}, m - v_l) + \\ & q_k h^{-lk}(\underline{V}, m - v_k) + q_l q_k h^{-lk}(\underline{V}, m - v_l - v_k) \Big] \end{aligned} \tag{10}$$

Similarly, we get

$$G\alpha_s(\underline{P},\underline{V}') = \sum_{m=s}^{L}\Big[h^{-lk}(\underline{V},m) + q_k h^{-lk}(\underline{V},m-v_l) + q_l h^{-lk}(\underline{V},m-v_k) + q_l q_k h^{-lk}(\underline{V},m-v_l-v_k)\Big] \quad (11)$$

Therefore we have

$$G\left(\alpha_s(\underline{P},\underline{V}) - \alpha_s(\underline{P},\underline{V}')\right) = (q_k - q_l)\sum_{m=s}^{L}\Big[h^{-lk}(\underline{V},m-v_k) - h^{-lk}(\underline{V},m-v_l)\Big] \quad (12)$$

If $p_l \le p_k$ and $q_i = \frac{p_i}{1-p_i}$, we also have $q_l \le q_k$. To prove the lemma, it is thus sufficient to show that

$$\sum_{m=s}^{L}\Big[h^{-lk}(\underline{V},m-v_k) - h^{-lk}(\underline{V},m-v_l)\Big] \ge 0 \quad (13)$$

The left hand side of (13) can be written as

$$\sum_{m=s-v_k}^{L-v_k} h^{-lk}(\underline{V},m) - \sum_{m=s-v_l}^{L-v_l} h^{-lk}(\underline{V},m) \quad (14)$$

Since $v_l \le v_k$, then also $s - v_l \ge s - v_k$ and $L - v_l \ge L - v_k$. Cancel out the identical terms in (14) and we thus get (14) equal to

$$\sum_{m=s-v_k}^{s-v_l-1} h^{-lk}(\underline{V},m) - \sum_{m=L-v_k+1}^{L-v_l} h^{-lk}(\underline{V},m) \quad (15)$$

Note that $h^{-lk}(\underline{V},m) \ge 0$ for all m and $h^{-lk}(\underline{V},m) = 0$ for $m > L - v_l - v_k$ (because when copies j and k are removed, the system is left with $L - v_l - v_k$ votes.) The right sum of (15) is equal to 0 and hence (15) is greater than or equal to 0.

References

[AA87] M. Ahamad and M.H. Ammar. Performance of quorum-consensus algorithms for replicated data. In *Sixth Symposium on Reliability in Distributed Software and Database Systems*, March 1987.

[AA88] D. Agrawal and A. El Abbadi. Reducing storage for quorum consensus algorithms. In *Very Large Databases*, 1988.

[AT86] A. El Abbadi and S. Toueg. Maintaining availability in partitioned replicated databases. In *Symposium on Principles of Database Systems (PODS) 1986*, pages 240–351, ACM, 1986.

[BG87] D. Barbara and H. Garcia-Molina. The reliability of voting mechanisms. *IEEE Transactions on Computers*, C-36(10):1197–1208, Oct 1987.

[BGS86] D. Barbara, H. Garcia-Molina, and A. Spauster. Protocols for dynamic vote reassignment. In *Principles of Distributed Computing*, pages 195 –205, ACM, 1986.

[Buz73] J. Buzen. Computational algorithms for closed queueing networks with exponential servers. *Commun. ACM*, 16(9):527–531, Sept 1973.

[DB85] D. Davcev and W.A. Burkhard. Consistency and recovery control for replicated data. In *Proceedings of 10th Symposium on Operating Systems Principles*, pages 87 – 96, ACM, Dec 1985.

[DGS85] Susan B. Davidson, Hector Garcia-Molina, and Dale Skeen. Consistency in partitioned network. *ACM Computing Survey*, 341 — 370, 1985.

[ES83] D.L. Eager and K.C. Sevcik. Achieving robustness in distributed database systems. *ACM Transactions on Database Systems*, 8(3):354–381, 1983.

[GB85] H. Garcia-Molina and D. Barbara. How to assign votes in a distributed system. *Journal of ACM*, 32(43):841–860, Oct 1985.

[Gif79] H. Gifford. Weighted voting for replicated data. In *Proceedings of 7th Symposium on Operating Systems (Pacific Grove, California)*, pages 150 –162, ACM, Dec 1979.

[Her87] M. Herlihy. *Dynamic Quorum Adjustment for Partitioned Data.* Technical Report CMU-CS-86-147, Carnegie-Mellon University, Pittsburgh PA 15213., 1987.

[JM87] S. Jajodia and D. Mutchler. Dynamic voting. In *SIGMOD-87 (San Francisco)*, ACM, 1987.

[Lam78] Leslie Lamport. The implementation of reliable distributed multiprocess systems. *Computer Networks*, 95–114, May 1978.

[Mur83] Katta G. Murty. *Linear Programming.* John Wiley and Sons, 1983.

[Par86] J.F. Paris. Voting with witnesses: a consistency scheme for replicated files. In *Proceedings of the Sixth International Conference on Distributed Computing*, pages 606–612, May 1986.

[Rot84] Joseph J. Rotman. *An Introduction to the Theory of Groups.* Allyn and Bacon, Inc., 1984.

[Tho79] Robert H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Transactions on Database Systems*, 4(2):180–209, June 1979.

[VD68] A. F. Veinott and G. B. Dantzig. Integral extreme points. *SIAM Review*, 10(3):371–372, July 1968.

Index	Read Set	Vote assignment				r
		v_1	v_2	v_3	v_4	
1	{{4}}	0	0	0	1	1
2	{{3}, {4}}	0	0	1	1	1
3	{{34}}	0	0	1	1	2
4	{{2}, {3}, {4}}	0	1	1	1	1
5	{{23}, {24}, {34}}	0	1	1	1	2
6	{{234}}	0	1	1	1	3
7	{{4}, {23}}	0	1	1	2	2
8	{{24}, {34}}	0	1	1	2	3
9	{{1}, {2}, {3}, {4}}	1	1	1	1	1
10	{{12}, {13}, {14}, {23}, {24}, {34}}	1	1	1	1	2
11	{{123}, {124}, {134}, {234}}	1	1	1	1	3
12	{{1234}}	1	1	1	1	4
13	{{4}, {12}, {13}, {23}}	1	1	1	2	2
14	{{14}, {24}, {34}, {123}}	1	1	1	2	3
15	{{124}, {134}, {234}}	1	1	1	2	4
16	{{4}, {123}}	1	1	1	3	3
17	{{14}, {24}, {34}}	1	1	1	3	4
18	{{3}, {4}, {12}}	1	1	2	2	2
19	{{13}, {14}, {23}, {24}, {34}}	1	1	2	2	3
20	{{34}, {123}, {124}}	1	1	2	2	4
21	{{134}, {234}}	1	1	2	2	5
22	{{4}, {13}, {23}}	1	1	2	3	3
23	{{34}, {124}}	1	1	2	3	5
24	{{14}, {23}, {24}, {34}}	1	2	2	3	4
25	{{24}, {34}, {123}}	1	2	2	3	5

Table 1: Read sets of 4 nodes

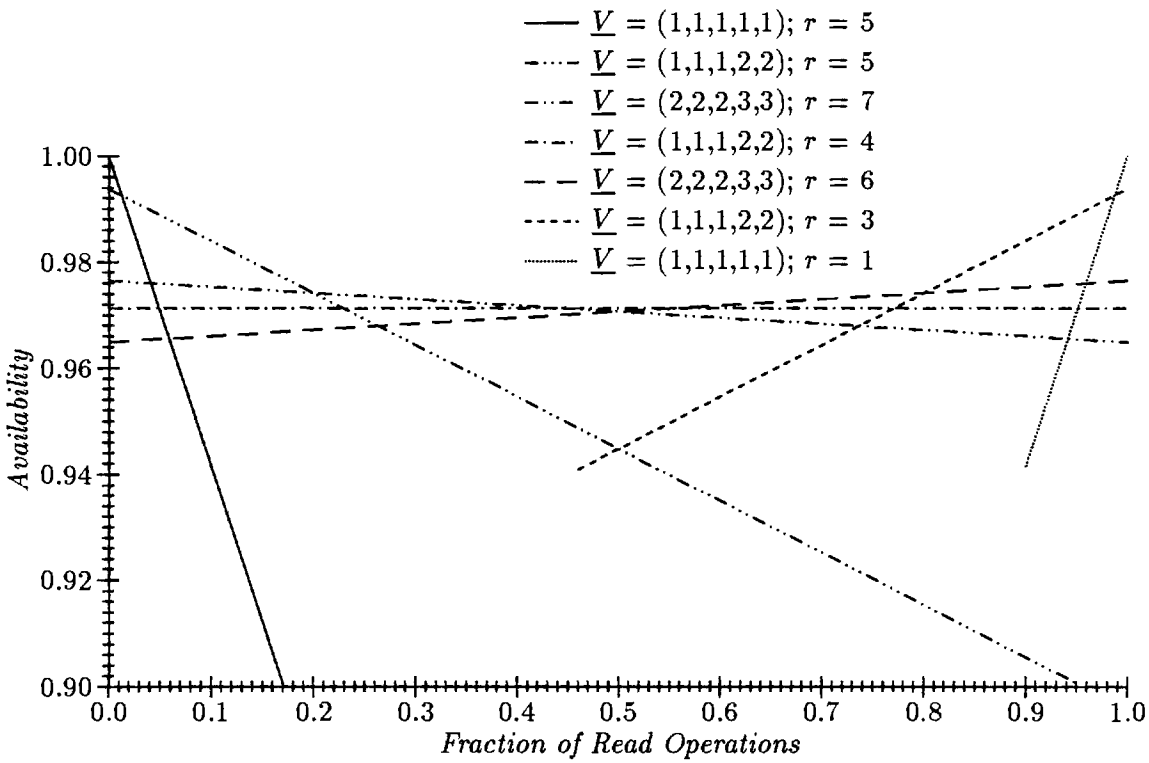


Figure 4: Read/write availability of read sets, $p_1 = p_2 = p_3 = 0.8$, other nodes = 0.9

	$p_5 = 0.9$, others = 0.8		$p_4 = p_5 = 0.9$, others = 0.8	
f	Vote assignment	r	Vote assignment	r
0.001	(1, 1, 1, 1, 1)	5	(1, 1, 1, 1, 1)	5
0.1	(1, 1, 1, 1, 1)	4	(1, 1, 1, 2, 2)	5
0.2	(1, 1, 1, 1, 2)	4	(1, 1, 1, 2, 2)	5
0.3	(1, 1, 1, 1, 2)	4	(2, 2, 2, 3, 3)	7
0.4	(1, 1, 1, 1, 1)	3	(2, 2, 2, 3, 3)	7
0.5	(1, 1, 1, 1, 1)	3	(1, 1, 1, 2, 2)	4
0.6	(1, 1, 1, 1, 1)	3	(2, 2, 2, 3, 3)	6
0.7	(1, 1, 1, 1, 2)	3	(2, 2, 2, 3, 3)	6
0.8	(1, 1, 1, 1, 2)	3	(1, 1, 1, 2, 2)	3
0.9	(1, 1, 1, 1, 1)	2	(1, 1, 1, 2, 2)	3
0.999	(1, 1, 1, 1, 1)	1	(1, 1, 1, 1, 1)	1

Table 2: Best read/write sets for two systems of 5 nodes

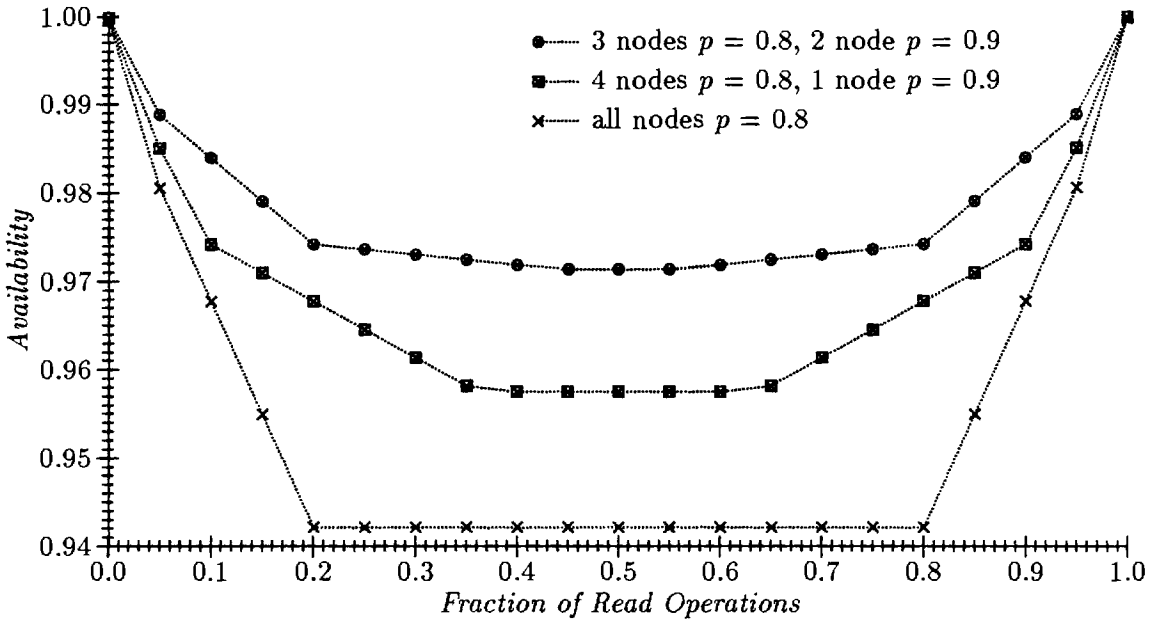


Figure 5: Optimal read/write availability for 3 systems of 5 nodes

f	Vote assignment	r
0.001	(1, 1, 1, 2, 2, 3, 3)	11
0.1	(1, 2, 2, 3, 4, 5, 7)	15
0.2	(3, 4, 5, 6, 8,10,13)	28
0.3	(2, 3, 4, 5, 6, 8,10)	21
0.4	(2, 2, 3, 4, 5, 6, 8)	16
0.5	(1, 2, 3, 3, 4, 5, 7)	13
0.6	(2, 2, 3, 4, 5, 6, 8)	15
0.7	(2, 3, 4, 5, 6, 8,10)	18
0.8	(3, 4, 5, 6, 8,10,13)	22
0.9	(1, 2, 2, 3, 4, 5, 7)	10
0.999	(1, 1, 1, 2, 2, 3, 3)	3

Table 3: Best read/write sets for a system of 7 nodes with reliability vector = (0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95)

$\alpha^{(1)}$ = optimum availability over all vote assignments and quorums
$\alpha^{(2)}$ = availability of the read one/write all quorum
$\alpha^{(3)}$ = availability of the read majority/write majority quorum
$\alpha^{(4)}$ = optimum availability using the uniform vote assignment

	All Vote Assignments			Uniform Vote Assignment			
f	Optimal $\underline{V}_N, r$		$\alpha^{(1)}$	$\alpha^{(2)}$	$\alpha^{(3)}$	r_{opt}	$\alpha^{(4)}$
0.001	(1, 1, 1, 1, 1, 1, 1)	7	0.999	0.064	0.866	7	0.999
0.1	(0, 0, 0, 0, 0, 1, 1)	2	0.972	0.157	0.866	5	0.937
0.2	(1, 1, 1, 1, 1, 5, 5)	10	0.955	0.250	0.866	5	0.901
0.3	(1, 1, 1, 1, 1, 4, 4)	8	0.943	0.344	0.866	4	0.866
0.4	(1, 1, 1, 1, 1, 3, 3)	6	0.933	0.438	0.866	4	0.866
0.5	(1, 1, 1, 1, 1, 3, 3)	6	0.933	0.531	0.866	4	0.866
0.6	(1, 1, 1, 1, 1, 3, 3)	6	0.933	0.625	0.866	4	0.866
0.7	(1, 1, 1, 1, 1, 4, 4)	6	0.943	0.719	0.866	4	0.866
0.8	(1, 1, 1, 1, 1, 5, 5)	6	0.955	0.813	0.866	3	0.901
0.9	(0, 0, 0, 0, 0, 1, 1)	1	0.972	0.906	0.866	3	0.937
0.999	(1, 1, 1, 1, 1, 1, 1)	1	0.999	0.999	0.866	1	0.999

Table 4: Optimizing Vote and Quorum versus Optimizing Quorum using Uniform Vote Assignment for a System of 7 nodes with Reliability Vector (0.6, 0.6, 0.6, 0.6, 0.6, 0.9 0.9)

NATIONAL SCIENCE FOUNDATION
Washington, D.C. 20550

FINAL PROJECT REPORT
NSF FORM 98A

PLEASE READ INSTRUCTIONS ON REVERSE BEFORE COMPLETING

PART I—PROJECT IDENTIFICATION INFORMATION

1. Institution and Address	2. NSF Program	3. NSF Award Number
Georgia Institute of Technology Office of Contract Administration Atlanta, GA 30332-0420	Networking & Comm. Res.	NCR-8604850
	4. Award Period From 12/87 To 11/90	5. Cumulative Award Amount $99,360.00

6. Project Title

Performance Analysis & Large Scale Information Systems

PART II—SUMMARY OF COMPLETED PROJECT *(FOR PUBLIC USE)*

Our work was geared towards the design and performance analysis of large scale information systems. The major problem behind such systems is how to provide a high capacity system. Our main work considers two basic approaches: the use of broadcast delivery and data replication. Since the use of broadcast delivery was conceptually well understood we concentrated on the practical aspects of the design and implementation of the design of a prototype

Our interest in data replication was in how to design a system that provides good response time characteristics.

We found, however, a few unanswered questions about how to optimally design a replicated data system. After addressing those problems, we went on to design the Grid Protocol, which provides a high performance database system. Another problem we addressed is how to collect responses in such a system which led us to an interesting generalization of the multiple access problem.

In consideration of the design of the information system over wireless packet radio networks, we determined that broadcast communication is best achieved via the use of TDMA protocols and proceeded to improve the understanding of such protocols.

Finally, we addressed the problem of how to find the location of a resource (e.g., a database) in a computer network when the resource is only known by name. We consider the use of a "generalized polling" protocol over a multiple access channel and the use of a "serial search" technique over a store and forward network.

PART III—TECHNICAL INFORMATION *(FOR PROGRAM MANAGEMENT USES)*

1. ITEM *(Check appropriate blocks)*	NONE	ATTACHED	PREVIOUSLY FURNISHED	TO BE FURNISHED SEPARATELY TO PROGRAM	
				Check (✓)	Approx. Date
a. Abstracts of Theses		x			
b. Publication Citations	x				
c. Data on Scientific Collaborators	x				
d. Information on Inventions	x				
e. Technical Description of Project and Results		x			
f. Other *(specify)* Selected papers		x			

2. Principal Investigator/Project Director Name *(Typed)*	3. Principal Investigator/Project Director Signature	4. Date
Mostafa Ammar		3/1/91

SF Form 98A (1-87) Supersedes All Previous Editions

Form Approved OMB No. 3145-0058

PART IV - SUMMARY DATA ON PROJECT PERSONNEL

NSF Division Networking & Comm. Dep.

The data requested below will be used to develop a statistical profile on the personnel supported through NSF grants. The information on this part is solicited under the authority of the National Science Foundation Act of 1950, as amended. All information provided will be treated as confidential and will be safeguarded in accordance with the provisions of the Privacy Act of 1974. NSF requires that a single copy of this part be submitted with each Final Project Report (NSF Form 98A); however, submission of the requested information is not mandatory and is not a precondition of future awards. If you do not wish to submit this information, please check this box ☐

Please enter the numbers of individuals supported under this NSF grant.
Do not enter information for individuals working less than 40 hours in any calendar year.

*U.S. Citizens/ Permanent Visa	PI's/PD's		Post-doctorals		Graduate Students		Under-graduates		Precollege Teachers		Others	
	Male	Fem.	Male	Fem.	Male	Fem.	Male	Fem.	Male	Fem.	Male	Fem.
American Indian or Alaskan Native												
Asian or Pacific Islander												
Black, Not of Hispanic Origin												
Hispanic												
White, Not of Hispanic Origin	1				3							
Total U.S. Citizens	1				3							
Non U.S. Citizens					2							
Total U.S. & Non-U.S. ..	1				5							
Number of individuals who have a handicap that limits a major life activity.												

*Use the category that best describes person's ethnic/racial status. (If more than one category applies, use the one category that most closely reflects the person's recognition in the community.)

AMERICAN INDIAN OR ALASKAN NATIVE: A person having origins in any of the original peoples of North America, and who maintains cultural identification through tribal affiliation or community recognition.

ASIAN OR PACIFIC ISLANDER: A person having origins in any of the original peoples of the Far East, Southeast Asia, the Indian subcontinent, or the Pacific Islands. This area includes, for example, China, India, Japan, Korea, the Philippine Islands and Samoa.

BLACK, NOT OF HISPANIC ORIGIN: A person having origins in any of the black racial groups of Africa.

HISPANIC: A person of Mexican, Puerto Rican, Cuban, Central or South American or other Spanish culture or origin, regardless of race.

WHITE, NOT OF HISPANIC ORIGIN: A person having origins in any of the original peoples of Europe, North Africa or the Middle East.

THIS PART WILL BE PHYSICALLY SEPARATED FROM THE FINAL PROJECT REPORT AND USED AS A COMPUTER SOURCE DOCUMENT. DO NOT DUPLICATE IT ON THE REVERSE OF ANY OTHER PART OF THE FINAL REPORT.

Performance Analysis of Large Scale Information Systems

Final Report for NSF Project NCR-8604550

Mostafa H. Ammar
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
Tel: (404)894-3292
E-mail: ammar@cc.gatech.edu

This final report discusses work that was carried out with the funding provided by NSF under grant number NCR-8604850.

Our work was geared towards the design and performance analysis of large scale information systems. The major problem behind such systems is how to provide a high capacity system. Our main work considers two basic approaches: the use of broadcast delivery and data replication. Since the use of broadcast delivery was conceptually well understood we concentrated on the practical aspects of the design and implementation of the design of a prototype. Our interest in data replication was in how to design a system that provides good response time characteristics. We found, however, a few unanswered questions about how to optimally design a replicated data system. After addressing those problems, we went on to design the Grid Protocol, which provides a high performance database system. Another problem we addressed is how to collect responses in such a system which led us to an intersting generalization of the multiple access problem. In our consideration of the design of information systems over wireless packet radio networks, we determined that broadcast communication is best achieved via the use of TDMA protocols and proceeded to improve the understanding of such protocols. Finally, we addressed the problem of how to find the location of a resource (e.g., a database) in a computer network when the resource is only known by name. We consider the use of a "generalized polling" protocol over a multiple access channel and the use of a "serial search" technique over a store and forward network.

The NSF grant supported several Ph.D. and M.S. students. Two students, Jose Bernabeu and Shun-Yan Cheung received their degrees in December 1988 and August 1990, respectively. Dr. Cheung is currently an assistant professor in the Department of Mathematics and Computer Science at Emory University, Atlanta. The abstracts form these two thesis are enclosed.

We briefly describe our research efforts next. Following this description is a list of publications that were supported by the NSF grant. A set of selected publications are also enclosed. The cover pages from all others are also enclosed.

Information Systems Using Broadcast Delivery
(Papers 14 and 15)

One of the challenges faced by an information system designer is how to configure a low cost system that can support a potentially large user population and still provide good response time. In typical systems the response time increases quickly with load. In this project we consider how to improve the response time performance of an information delivery system by exploiting the inevitable commonality of information need present in a large user base. We use the broadcast delivery of responses in conjunction with a design that allows the response to a user's request to satisfy other requests and thus improving response time and reducing the load experienced by the information server.

Prototyping a Broadcast Delivery Information System A system using the above ideas has been designed and is currently being implemented using the facilities of the Telecommuni-

cations Laboratory. The system is being developed using Sun SPARC stations as the client and information server hardware and Ethernet as the broadcast network. The system has been under development for about nine months and is currently operational in a limited way. Once fully operational, we intend to make the system available for "beta" testing by selected users in our department.

Scheduling algorithms for broadcast information delivery systems An important operational strategy of information systems using broadcast delivery is the scheduling algorithm used to select the next request for processing. In this study we formulate the scheduling problem and attempt an exact optimization procedure. Based on that experience several scheduling heuristics are identified and evaluated. The most promising of these heuristic are slated for implementation in our prototype above.

Data Replication
(Papers 1, 2, 7, 9, 13 and 16)

Fault tolerance in distributed database systems can be achieved by replicated data at nodes with independent failure modes. When data is replicated, algorithms known as *replica control protocols* must be used to maintain the consistency of the copies of the data. We consider protocols that are based on quorum consensus where an operation (read or write) may proceed only if it obtains permission from nodes that constitute a quorum group.

Our work in this area has consisted of the following contributions:

1. Exploration of how the parameters of quorum consensus protocols can be chosen so that the performance of the protocol are optimized.
2. The development of a high performance (low response time) replica control protocol known as the Grid protocol.
3. Consideration of the performance of quorum consensus protocol from the user's viewpoint and how that is affected by the network topology and the placement of data copies in the network.
4. The development of the Multi-Dimensional (MD) Voting technique which unifies the various static quorum consensus techniques.

Resource Finding in Computer Networks
(papers 10, 11, an 12.)

In this work we address the problem of finding the location of a resource in a computer network when the resource is only known by name. We investigate techniques using multicast communication. We consider the use of a *generalized polling* protocol in which nodes are

queried for their knowledge of a resource's whereabouts in groups. We describe a procedure to determine the optimal subdivision of nodes into multicast groups.

In another piece of work we investigate the use of *Serial Search* as a search technique in store-and-forward networks. In this technique, nodes in the network are visited serially until a node that possesses information about the resource's whereabouts is found. We analyze the cost of such a procedure and develop an algorithm to determine the optimal serial search in a tree network.

Generalizing the Multiple Access Problem
(Paper 6.)

We consider a generalization of the multiple access problem where it is necessary to identify a subset of ready users, not all. The problem is motivated by several "response collection" applications that arise in distributed computing and database systems. In these applications, a collector is interested in gathering a set of responses from a number of potential respondents. The collector and respondents communicate over a shared channel. We define some collection objectives and investigate the performance of a suite of protocols that can be used to achieve these objectives. The protocols are based on the use of polling, TDMA, and group testing. Our concern is with cost measures that reflect the computational load placed on the system, as well as the delay incurred for achieving a particular objective.

TDMA Protocols for Packet Radio Networks
(Papers 5 and 8.)

Time Division Multiple Access (TDMA) protocols provide packet radio networks with two features that facilitate efficient communications. First, they eliminate the possibility of collisions. Second, they allow for the spatial reuse of the radio channel bandwidth by permitting more than one node to transmit at once. Many different algorithms have been proposed to maximize the reuse of the bandwidth and simultaneously minimize the transmission cycle length. The resulting slot assignments can be grouped into two general general strategies – *Node* and *Link* allocation. Node allocation involves assigning a node a timeslot during which it may transmit to any of its neighbors. A link allocation scheme allocates unique to a node for each directed link it has to a neighbor.

In our first piece of work we evaluate the performance of the two slot assignment strategies using a detailed simulation model. We consider networks carrying single destination as well as broadcast traffic. Our conclusion is that the Node Allocation strategy consistently outperforms the Link Allocation strategy. In the next piece of work we address the problem of TDMA scheduling in a mobile network environment. We develop a procedure that allows a node to move and reallocate itself a transmission slot without involving the entire network. The procedure uses an *optimistic* approach and defines recovery procedures tha can be used when inconsistent TDMA schedules are detected.

Publications Supported by NSF Grant

1. Ahamad, M. , Ammar, M. H., Cheung, S. Y., "Multi-Dimensional Voting," to appear in the *ACM Transactions on Computer Systems.*

2. Cheung, S. Y., Ammar, M. H., Ahamad, M., "The Grid Protocol: A High Performance Scheme for Maintaining Replicated Data," to appear in *IEEE Transactions on Knowledge and Data Engineering.* (also in Proc. of the Sixth IEEE International Conference on Data Engineering, 1990.)

3. ** Cheung, S.Y., Ahamad, M., Ammar, M. H., "Optimizing Vote and Quorum Assignments for Reading and Writing Replicated Data," *IEEE Transactions on Knowledge and Data Engineering,* Vol. 1, # 3, September 1989, pp 387-397. (also in Proc. of Fifth International Conference on Data Engineering, 1989.)

4. Ammar, M. H., Stevens, D. S., "A Distributed TDMA Rescheduling Procedure for Mobile Packet Radio Networks," to appear in Proceedings of the IEEE International Conference on Communications, Denver, Colorado, June 1991.

5. ** Ammar, M. H., Rouskas, G., "On the Performance of Protocols for Collecting Responses Over a Multiple Access Channel," to appear in Proceedings of IEEE INFOCOM '91, Miami, Florida, April 1991. (submitted to *IEEE Transactions on Communications.*)

6. Ahamad, M., Ammar, M. H., Cheung, S.Y., "Optimizing the Performance of Replica Control Protocols," Proceedings of the IEEE Workshop on the Management of Replicated Data, Houston, Texas, November 1990, pp102-107.

7. Ammar, M. H., Stevens, D.S., "Evaluation of Slot Allocation Strategies for TDMA Protocols in Packet Radio Networks," Proceedings of The 1990 Military Communications Conference (MILCOM), Monterey, California, October 1990, pp835-839.

8. Ammar, M. H., Ahamad, M., Cheung, S. Y., "Performance of Quorum Consensus Protocols for Mutual Exclusion from the User's Point of View," Proceedings of the Second IEEE Workshop on Future Trends of Distributed Computing Systems, Cairo, Egypt, September 1990, pp413-419. (submitted to *IEEE Transactions on Reliability*)

9. ** Bernabeu, J., Ahamad, M., Ammar, M. H., "Resource Finding in Store-and-Forward Networks," Proceedings of INFOCOM 90, San Fransisco, California, June 1990, pp819-826. (submitted to *ACTA Informatica*)

10. M. Ahamad, M. H. Ammar, J. M. Bernabeu and M. Y. Khalidi, "Using Multicast Communication to Locate Resources in a LAN-based Distributed System," Proc. of the 13th IEEE Conference on Local Computer Networks, October, 1988, Minneapolis, Minnesota, pp193-202.

11. J. M. Bernabeu, M. H. Ammar and M. Ahamad, "Optimal Multicast Groupings for Locating Resources in Distributed Systems," in Proc. of IEEE INFOCOM, April 1989, Ottawa Ontario, Canada, pp312-321. revised version submitted to *Computer Networks and ISDN Systems*)

12. ** S. Y. Cheung, M. Ahamad and M. H. Ammar, "Multi-dimensional Voting: A General method for Implementing Synchronization in Distributed Systems," Proc. of the 10th International Conference on Distributed Computing Systems, Paris, France, June 1990, pp362-369.

13. S. Y. Cheung, M. H. Ammar, M. Ahamad, "On the Optimality of Voting," Georgia Tech Technical Report GIT-ICS-90/30, 1990. (submitted to *Information Processing Letters*)

14. ** M. H. Ammar, H. J. Kim, "Prototyping a Broadcast Delivery Information System," Georgia Tech Technical Report GIT-ICS-90/16, March 1990.

15. H. D. Dyekman, J. W. Wong and M. H. Ammar, "Scheduling Algorithms for Broadcast Delivery Information Systems," Georgia Tech Technical Report GIT-ICS-88/35, October 1988

16. M. Ahamad, M. H. Ammar, S.Y. Cheung, "Replicated Data Management in Distributed Systems," Submitted to *IEEE Computer Magazine.*

** Publications enclosed in full.

Thesis Abstracts

Location Finding Algorithms for Distributed Systems

a Thesis
Presented to
The Faculty of the Division of Graduate Studies

By

José Manuel Bernabéu Aubán

In Partial Fulfillment
of the Requirements for the Degree of
Doctor of Philosophy
in the School of Information and Computer Science

Georgia Institute of Technology
December 1988

Summary

One of the problems encountered in distributed systems is how to find the location of the resources needed by a computation. In many situations the location may have to be found at run time, when the resource is accessed, thus the efficiency of the location algorithm will affect the performance of the system. In general, the larger the distributed system, the more the number of processors at which a resource may reside at the time it is accessed. The general problem of resource location in distributed systems has not been addressed adequately, and most of the systems have adopted ad hoc solutions without a careful study of the performance of the algorithms used. In this thesis it is studied the problem of finding the location of resources in order to get a better understanding of the factors affecting the cost of a location algorithm. This study will make it possible to judge proposed algorithms as well as to come up with new ones, optimized for particular systems.

Most distributed systems are based on bus networks that have broadcast and multicast capabilities. The thesis first describes an efficient location method that takes advantage of the multicast capabilities of these networks to reduce the computation cost of resource location finding. Performance results based on a simulation of the scheme are presented, showing that the method is a simple and efficient one. An approximate analysis is also presented, and it is shown that the analysis provides an extremely good approximation for low and high values of the load in the system. In another multicast scheme for broadcast networks, the thesis considers a system in which no references to resources are stored in the network except where the resource resides. Besides the CPU cost, response time costs are also considered, and a cost formula is found for the scheme. Based on this cost formula, an algorithm is presented to find an optimal sequence of multicast groups to be used in locating a resource.

The thesis then considers the communication costs incurred by location finding algorithms in store–and–forward networks. A model of such system is first constructed and, based on this model, a worst case analysis is performed to obtain a lower bound on the number of messages needed to locate a resource when no information about the location of the resource is available at the node conducting the search. It is also shown that when the searcher node has the probability distribution indicating the location of the resource in the system, the problem of finding the optimal way to traverse the network has only a polynomial time algorithm for restricted classes of networks.

The use of hint tables can reduce the cost of resource location when a resource is used repeatedly. The thesis presents a model of the usage of hint tables and shows how it affects the performance of finding the location of resources.

Optimizing the Performance of Quorum Consensus Replica Control Protocols

Shun Yan Cheung

133 pages

Directed by Dr. Mustaque Ahamad and Dr. Mostafa H. Ammar

This thesis considers the performance of synchronziation protocols based on *quorum consensus* in distributed systems. In these protocols, an operation can proceed if permission can be obtained from nodes that constitute a *quorum group*. The collection of all quorum groups is a *quorum set*. Voting can be used to define quorum sets and it is appealing because it is flexible and can be easily implemented. However, voting cannot be used to represent all quorum sets.

We first study the problem of optimizing the system availability of quorum consensus methods and presents a direct method for finding the optimal quorum set for mutual exclusion, and reading and writing of replicated data. We show that the optimal system availability can be achieved by voting.

The thesis then considers optimizing an arbitrary performance measure. Changes in the quorum set cause performance changes in a discrete and highly complex manner, and a direct method is difficult to obtain. However, when the quorum set is given, the system behavior is fixed and the performance can then be computed with relative ease. The thesis presents an efficient algorithm for generating the universe of vote assignable quorum sets.

The optimal voting parameter settings can be obtained by a search.

The thesis next presents a non-voting based quorum consensus protocol, called the Grid Protocol, that has small quorum groups. An analysis shows that the data availability of this protocol can be as high as voting and simulation results show that transactions using the grid protocol can have lower response time than voting.

Finally, the *multi-dimensional voting* concept is investigated where vote and quorum assignments are k-dimensional vectors of non-negative integers. Each dimension of the vote and quorum assignment is similar to voting and the quorum requirements in different dimensions can be combined in a number of ways. Multi-dimensional voting is as general as quorum sets but has the advantage that it is flexible and easy to implement. Several replica control protocols are implemented using multi-dimensional voting which illustrate the versatility of this technique.

Publication Cover Pages

Paper #1: To Appear in ACM Transactions on Computer Systems

Multi-Dimensional Voting*

Mustaque Ahamad†
Mostafa H. Ammar†
Shun Yan Cheung‡

†College of Computing
Georgia Institute of Technology, Atlanta, GA 30332

‡Department of Mathematics and Computer Science
Emory University, Atlanta, GA 30322

Abstract

We introduce a new concept, *multi-dimensional voting*, in which the vote and quorum assignments are k-dimensional vectors of non-negative integers and each dimension is independent of the others. Multi-dimensional voting is more powerful than traditional weighted voting because it is equivalent to the general method for achieving synchronization in distributed systems which is based on sets of groups of nodes (quorum sets). We describe an efficient algorithm for finding a multi-dimensional vote assignment for any given quorum set and show examples of its use. We demonstrate the versatility of multi-dimensional voting by using it to implement mutual exclusion in fault-tolerant distributed systems, and protocols for synchronizing access to fully and partially replicated data. These protocols cannot be implemented by traditional weighted voting. Also, the protocols based on multi-dimensional voting are easier to implement and/or provide greater flexibility than existing protocols for the same purpose. Finally, we present a generalization of the multi-dimensional voting scheme, called *nested multi-dimensional voting*, that can facilitate implementation of replica control protocols that use structured quorum sets.

*This work was supported in part by NSF grants NCR-8604850 and CCR-8806358.

Paper #2: To appear in IEEE Transactions on Knowledge and Data Engineering

The Grid Protocol: A High Performance Scheme for Maintaining Replicated Data*

Shun Yan Cheung
Mostafa H. Ammar
Mustaque Ahamad
Georgia Institute of Technology

GIT-ICS-89/22

June 15 , 1989

Abstract

We present a new protocol for maintaining replicated data that can provide both high data availability and low response time. In the protocol, the nodes are organized in a logical grid and interconnected by a network providing multicast facilities. Existing protocols are designed primarily to achieve high availability by updating a large fraction of the copies which provides some (although not significant) load sharing. In the new protocol, transaction processing is shared effectively among nodes storing copies of the data and both the response time experienced by transactions and the system throughput are improved significantly. We present an analysis of the availability of the new protocol and use simulation to study the effect of load sharing on the response time of transactions. We also compare the new protocol with a voting based scheme.

School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

*This work was supported in part by NSF grants NCR-8604850 and CCR-8806358.

Optimizing Vote and Quorum Assignments for Reading and Writing Replicated Data

SHUN YAN CHEUNG, MUSTAQUE AHAMAD, AND MOSTAFA H. AMMAR, MEMBER, IEEE

Abstract—In the weighted voting protocol which is used to maintain the consistency of replicated data, the availability of the data to read and write operations not only depends on the availability of the nodes storing the data but also on the vote and quorum assignments used. We consider the problem of determining the vote and quorum assignments that yield the best performance in a distributed system where node availabilities can be different and the mix of the read and write operations is arbitrary. The optimal vote and quorum assignments depend not only on the system parameters such as node availability and operation mix, but also on the performance measure. We present an enumeration algorithm that can be used to find the vote and quorum assignments that need to be considered for achieving optimal performance. When the performance measure is data availability, an analytical method is derived to evaluate it for any vote and quorum assignment. This method and the enumeration algorithm are used to find the optimal vote and quorum assignment for several systems. The enumeration algorithm can also be used to obtain the optimal performance when other measures are considered.

Index Terms—Availability, data replication, fault tolerance, replica control methods, vote and quorum assignment, weighted voting.

I. INTRODUCTION

A DISTRIBUTED system consists of a number of potentially unreliable nodes interconnected via a communication subnetwork. The resources stored at the nodes can be shared and when a node fails, the resources stored at the node become unavailable. Replicating resources at different nodes with independent failure modes can enhance availability and fault tolerance, since a resource could be available even when some nodes have failed. When data are replicated, care must be taken to preserve consistency among the various copies or *replicas*. In addition to increased availability, replication can also provide improved performance of read transactions by reducing the network communication cost since these transactions can access the data from the local replica.

A large number of replica control protocols have been developed to maintain the consistency of replicated data [1]. In this paper, we address the issue of optimization for a voting-based replica control protocol by deriving a general method for finding the optimal settings for the parameters of the protocol. We consider the voting mechanism because it has proven to be flexible and relatively easy to implement.

Voting has been used for various applications in distributed systems. In [2], Gifford proposed its use for synchronizing read and write operations on replicated files. Each file replica is assigned some number of votes and each operation is required to obtain a predefined quorum of votes to proceed. To ensure that a read operation returns the value installed by the last write operation, the read and write operations must acquire r and w number of votes, respectively, such that $r + w > L$, where L is the total number of votes assigned to all replicas. The values r and w are called the read and write quorum. Generally, $r + w = L + 1$ is used which ensures that each read quorum has a nonempty intersection with each write quorum. Since all replicas need not be updated when a write operation completes, timestamps or version numbers must be used in order to determine the value that is written most recently. When version numbers are used, each write quorum must also intersect with every other write quorum, i.e., $2w > L$ [2].

A number of replica control protocols have been derived from weighted voting. Eager and Sevcik introduced a dynamic scheme based on voting that allows the system to switch between normal and failure modes [3] (which have different values for read and write quorums). The system can also change the quorum assignment in the schemes presented in [4]–[6] and the vote assignment can be changed in the scheme described in [7]. Other protocols based on voting are presented in [8]–[10].

The problem of assigning votes to achieve mutual exclusion is addressed by Garcia-Molina and Barbara in [11]. When the quorum for each operation is a majority of all votes assigned, each operation will have mutually exclusive access to the data. In general, mutual exclusion can be guaranteed by defining a set of groups of nodes [12], called a coterie, such that any two groups in a coterie have a nonempty intersection. When voting is used, the groups of nodes that have a majority of the votes constitute a coterie (there exist coteries that cannot be obtained from any vote assignment [11]). In [11], it is shown that only a finite set of vote assignments need to be considered to get all coteries that can be obtained from vote assignments. Thus, it is not necessary to deal with the unbounded set of possible vote assignments. In another work, the same authors have considered the problem of

Manuscript received September 28, 1988; revised July 11, 1989. This work was supported in part by the National Science Foundation under Grants CCR-8806358 and NCR-8604850.

The authors are with the School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA 30332.

IEEE Log Number 8930638.

Paper #4: To appear [illegible]

A Distributed TDMA Rescheduling Procedure for Mobile Packet Radio Networks *

Mostafa H. Ammar *David S. Stevens*
College of Computing, Georgia Institute of Technology, Atlanta, Georgia 30332

Abstract

Packet radio networks provide two features not present in a wire-based network - mobility and a broadcast channel. The goal of maximizing the use of the bandwidth when using TDMA seems to conflict with the goal of allowing mobile nodes to *locally* reallocate themselves TDMA slots so that collisions will not occur. We present a procedure that permits a node to move and then reallocate itself a transmission slot without involving the entire network. In our procedures the channel over which control packets are exchanged is shared and unreliable. Therefore the resulting TDMA schedule may not be collision free. We present a collision resolution algorithm to correct these problems. Finally a procedure by which nodes can allocate themselves additional transmission slots, if they are available and which maximizes bandwidth utilization, is given.

1 Introduction

Packet radio networks have unique features that wire-based networks lack, that being mobility and a broadcast medium. Time Division Multiple Access (TDMA) protocols provide packet radio networks with collision free communications and permit spatial reuse of the radio channel by allowing more than one node to transmit at once. Many algorithms exist for allocating transmission rights to nodes in a packet radio network that produce a TDMA collision free schedule [1, 2, 3, 4, 5].

When a TDMA protocol is coupled with mobile nodes, established collision free schedules deteriorate. To accommodate mobile nodes, many distributed solutions [6, 7] require long TDMA frames and allocate only a single slot per node or directed link per TDMA frame, and thus may result in a schedule that does not fully use the available broadcast channel. Algorithms that allocate more than one slot to a node or directed link per TDMA frame [1, 2, 5, 8, 9] require the entire network become involved in the reallocation procedure when a node moves or a new node enters the network.

It appears that algorithms that address the maximal use of the bandwidth do not adequately handle topological changes. (By *maximal* we mean that an additional slot cannot be allocated to a node for transmission without causing collisions.) Those that can locally adapt their schedules to accommodate mobile nodes do not use all the available bandwidth. In this paper we describe a procedure for adjusting TDMA slot allocation to nodes in a mobile packet radio network where maximal use of the bandwidth is achieved, rearrangement of the TDMA schedule involves only a few nodes, and where control messages are transmitted over a shared error-prone channel. The procedure is an *optimistic* one in that nodes make a best effort at assigning themselves collision free slots. Special recovery procedures are defined in case inconsistencies causing collisions are discovered in the slot assignments.

The paper is organized as follows: Section 2 describes the packet radio network assumptions underlying our investigation. In Section 3 we outline and motivate our approach to solving the problem. In Section 4 we discuss the recovery procedures invoked when a node discovers inconsistencies in the TDMA slot assignments. Section 5 contains our collision resolution procedure. In Section 6 we present a distributed procedure that can be used by nodes to allocate slots to achieve maximal fair allocation of the TDMA frame. Some concluding remarks are given in Section 7.

2 Network Assumptions

First, we assume that if node u can hear node v, then v can also hear u. All packets are of a constant length, and we will partition time into constant size time slots that are equal to the packet length plus the maximum propagation delay. We assume that the network is initialized with a slot assignment algorithm that allocates at least one slot per node per TDMA frame during which it can transmit to any of its neighbors. A time slot is available if assigning it does not result in a collision. To avoid collisions, the following two conditions must hold:

- C1. Node v does not transmit in the same period during which a neighbor is transmitting to it. (A node cannot send and receive simultaneously.)
- C2. Only one neighbor of node v can transmit to it during any one period of time. (A node cannot simultaneously receive two transmissions.)

Finally, we assume that nodes periodically broadcast *status packets* to their neighbors. These packets will contain information about the slot assignments for this node, as well as its neighbors.

*M.H. Ammar is supported by NSF grant NCR-8604850 and Major D.S. Stevens is supported by the U.S. Army Institute for Research in Management Information, Communications and Computer Sciences (AIRMICS).

Submitted to IEEE Trans. on Comm.

On the Performance of Protocols for Collecting Responses over a Multiple-Access Channel *

Mostafa H. Ammar and George N. Rouskas
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332

Abstract

We consider a generalization of the multiple access problem where it is necessary to identify a subset of the ready users, not all. The problem is motivated by several "response collection" applications that arise in distributed computing and database systems. In these applications, a collector is interested in gathering a set of responses from a number of potential respondents. The collector and respondents communicate over a shared channel. We define three collection objectives and investigate a suite of protocols that can be used to achieve these objectives. The protocols are based on the use of polling, TDMA, and group testing. Using a binomial respondent model we analyze and, where applicable, optimize the performance of the protocols. Our concern is with cost measures that reflect the computational load placed on the system, as well as the delay incurred for achieving a particular objective.

1 Introduction

We investigate the problem of how to best collect a specified number of responses from a set of nodes over a multiple access channel. Several situations in distributed systems where such a problem arises are described later. We consider a system where nodes share a common communication channel. One node in the system is interested in collecting responses from the other nodes. Not all nodes can or will respond when requested and the node soliciting responses is interested in achieving a collection objective.

The problem we consider is actually a generalization of the multiple access communication problem where we are concerned with identifying a *subset* of ready users, not all. A response collection process will be aimed at achieving one of a set of *collection objectives* to be described later. We describe and analyze a suite of protocols that can be used for response collection. Our concern is with the cost of the collection process in terms of the amount of computation resources it consumes, as well as the amount of time expended to achieve a certain collection objective. The protocols we use are based on the use of polling, time division multiple access (TDMA) and group testing.

Whereas polling and TDMA are well known multiple access techniques, group testing warrants a short introduction. It is a technique that can be used to efficiently identify "defective" items in a set. It has been studied extensively in different contexts (see for example [1, 2, 3, 4]). The basic idea of the technique is the testing of items being inspected in groups. The composition of the group to be tested at any one point in time being dictated by the history of previous test outcomes. Each test is counted as a single step and the objective is to determine group composition rules to minimize the number of steps. In its original form, the problem assumes the outcome of each test would indicate one of two situations: "all items are not defective" or "there is at least one defective item." We are concerned here with the potential use of group testing as a technique for collision resolution over a multiple access channel. Such use has been described in [5, 6, 7, 8]. The additional feature when using group testing over a multiple-access channel is the ability to differentiate among three possible outcomes when a group is enabled: no transmission, one transmission, and more than one transmission (a collision).

This paper is organized as follows. In section 2 we discuss some applications that motivate our work. Section 3 contains a model of our system. Section 4 presents a description and analysis of the Polling and TDMA protocols. In section 5 we describe and analyze an approach based on the staging of the response collection process where in each stage a TDMA protocol is employed. Sections 6 and 7 investigate the group testing and staged group testing protocols. Some numerical examples are presented in section 8 and section 9 contains some concluding remarks.

2 Some Applications

The following are some applications that make use of response collection.

A database system with multiple query optimization: Here we have a shared channel LAN with the primary purpose of giving a set of attached users access to a database (also connected to the network). The users are moderately active and the database employs sophisticated query processing techniques. These include schemes to speedup query processing through the use of multiple query optimization (see e.g., [9]). Rather than processing each query individually, the database tries to process a number of queries (up to a maximum) at a time. In order to manage memory and processing resources efficiently, the database processor prefers to actively collect responses, rather than receiving responses asynchronously.

*This work is supported in part by NSF grant NCR-8604850

of Replicated Data.

Optimizing the Performance of Quorum Consensus Replica Control Protocols*

Mustaque Ahamad
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332

Mostafa H. Ammar
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332

Shun Yan Cheung†
Dept. of Math. & Comp. Science
Emory University
Atlanta, Georgia 30322

Abstract

We present in this paper a summary of the results of our research in replica control protocols that are based on quorum consensus. In quorum consensus methods, operations are required to obtain permission from a quorum group of nodes to proceed to completion and the collection of quorum groups is called a quorum set. In the summary we present the techniques that we have developed for finding the quorum set that maximizes a given performance measure. We also present a brief discussion of the optimality of voting, a replica control protocol that can effectively reduce response time through load sharing, and the *multi-dimensional voting* (MD) technique, that can be used to define all quorum sets. An MD-voting based implementation of a dynamic quorum consensus protocol that allows the synchronization procedure to adapt to the current state of the system is also presented.

1 Introduction

Distributed systems offer many advantages including fault-tolerance which can be achieved by replicating resources at nodes with independent failure modes. When data (e.g., files) is replicated, algorithms must be used to maintain the consistency of the copies or *replicas* of the data. Such algorithms, called *replica control protocols*, implement rules for accessing the replicas to ensure correctness (e.g., single-copy serializability). A large number of replica control protocols have been proposed in the literature. These include voting, available copies, primary copy and many others. The main focus of these protocols has been to enhance *availability* by tolerating as many node and communication failures as possible. Availability can be defined as the steady state probability that a transaction is able to access the data successfully when it arrives to the system.

Data replication can also be used to improve other performance measures. For example, the execution of a transaction requires reading of data from disk, processing and possibly writing the data to the disk (when it is modified). If data is not replicated, all transactions that access data stored at a node must wait for the data to be read or written. When the data is replicated, load generated by the requests can be shared by nodes having the replicas and hence the response time of the transactions can be improved. Notice that the degree of sharing depends on the replica control protocol used. If read transactions can access any replica (a write transaction must update all replicas to ensure correctness), the load generated at each node by the read transactions will be $1/n$ compared to when no replication is used (n is the number of replicas).

We consider protocols that are based on quorum consensus [1]. An operation proceeds to completion only if it can obtain permission from nodes that constitute a *quorum group* [2]. Quorum groups used by conflicting operations have non-empty intersections to guarantee proper synchronization. The collection of quorum groups used by an operation is known as a *quorum set*. If each group in the quorum set intersects with every other group in the set, it is called a *coterie* [3] and it can be used to achieve mutual exclusion. Weighted voting [4] is a representation technique to define quorum sets so that quorum groups need not be listed explicitly. It is shown in [3] that there exist quorum sets that cannot be defined by voting.

We will summarize the results of our research in Sections 2–6 and they include techniques for finding the quorum set that optimizes a given performance measure, a replica control protocol for reducing response time, the multi-dimensional voting concept and an implementation of a dynamic replica control method.

2 Optimal System Availability

2.1 Homogeneous Systems

We have explored how optimal vote and quorum assignments can be obtained for a system for read and write transactions when their mix could be arbitrary. In [5], we considered the problem in a system where node reliabilities are identical. The performance measures considered are the *system* availability (i.e., the probability that some part of the system is available) to transactions without blocking (a transaction aborts instantaneously when the currently operational nodes do not have sufficient votes to form a desired quorum) and the mean response time when transactions wait for nodes to recover from failures until a quorum is available. One of the interesting results shows

*This work was supported in part by NSF grants NCR-8604850 and CCR-8806358.

†Work was done while this author was at the College of Computing, Georgia Institute of Technology, Atlanta, Georgia.

(MILCOM '90)

Evaluation of Slot Allocation Strategies for TDMA Protocols in Packet Radio Networks*

David S. Stevens *Mostafa H. Ammar*
College of Computing, Georgia Institute of Technology, Atlanta, Georgia 30332

Abstract

Time Division Multiple Access (TDMA) protocols provide packet radio networks with useful features that facilitate efficient communications. Many different algorithms have been proposed to maximize the reuse of the bandwidth and simultaneously minimize the TDMA frame length. The resulting slot assignments of these algorithms can be grouped into two general strategies for assigning transmissions rights to nodes - *Node* and *Link* Allocation. In this paper the performance of each allocation strategy is evaluated using a detailed simulation. Our results indicate that in all cases for both single destination packets and broadcast packets node allocation offers better delay performance than link allocation.

1 Introduction

A Packet radio network (PRNET) offers two unique features that a cable connected store-and-forward network lacks - a broadcast medium and a dynamic topology. The broadcast medium permits any network node within range of the transmitting node to receive the packet. The use of a radio channel instead of wires to link the network nodes also permits them to move about freely. Yet these same features are the source of the major challenges to efficient protocol design. The broadcast medium limits the number of nodes that can successfully transmit at the same time while mobile nodes can create problems for deterministic channel access protocols.

A Time Division Multiple Access (TDMA) scheme makes a considerable effort at maximizing the spatial reuse of the available bandwidth while simultaneously eliminating the possibility of collisions. This facilitates the delivery of packets throughout the network. TDMA protocols can be grouped together by how they divide the channel bandwidth among the network members. Two common schemes are used and we refer to these two techniques as *node* allocation and *link* allocation. For static networks different allocation algorithms have been proposed to minimize the TDMA frame length while maximizing the spatial reuse of the bandwidth [1, 2, 3, 4]. These are centralized slot assignment algorithms that require global knowledge of the network topology. A distributed node allocation algorithm has been proposed in [5] for dynamic networks and a link allocation algorithm has been presented in [6]. Both algorithms use greedy selection heuristics to create an initial collision free environment.

Although packet radio network protocols are designed to carry primarily single destination packets, point-to-multipoint communication is required for many applications. Recently, the issues involving the delivery of packets addressed to multiple destinations in a packet radio network have received attention, with the emphasis being on broadcasting [6, 7, 8, 9, 10]. Many of these protocols for broadcasting require the use of a TDMA channel access scheme. Yet, the performance of broadcast communication in packet radio networks using a TDMA channel access protocol has not been investigated. We consider networks carrying single destination packets as well as broadcast packets.

For a packet radio network using a TDMA channel access scheme, it is not clear which time slot allocation strategy - node or link, will provide the better delay performance. In this paper we examine the delay experienced by single destination and broadcast packets in networks using link and node allocation strategies. As mathematical analysis of arbitrary topologies is not feasible, we conduct our investigation using a detailed simulation. Our results indicate that in all cases node allocation offers better delay performance than link allocation.

In the section that follows we describe our version of the link and node allocation schemes and discuss their attributes. Section 3 describes our simulation model. In Section 4 we discuss the results obtained using our simulation for networks carrying only single destination packets. In Section 5 we discuss the delay performance of two protocols: multidestination routing and controlled flooding, that can be used to route broadcast packets in packet radio networks using TDMA. Section 6 contains some concluding remarks.

2 Node and Link Allocation Strategies

In this section we discuss the node and link slot allocation schemes. We then examine the implications of the TDMA frame length and discuss the advantages and disadvantages of each allocation method.

*M.H Ammar is supported by NSF grant NCR-8604850 and Major D.S. Stevens is supported by the U.S. Army's fully funded graduate program.

Paper # 6
in Distributed Computing Systems, 1990
Submitted to IEEE Transactions on Reliability

Performance of Quorum Consensus Protocols for Mutual Exclusion from the User's Point of View*

Mostafa H. Ammar *Mustaque Ahamad* *Shun Yan Cheung*
College of Computing, Georgia Institute of Technology, Atlanta, Georgia 30332

Abstract

A general class of protocols used for achieving mutual exclusion in distributed systems is quorum consensus. In these methods, an operation must obtain permission from a group of coordinators before it can proceed to completion. We consider a store-and-forward network with coordinators resident in some of the switching nodes. The main motivation for having multiple coordinators is to enhance system availability. Most studies of the availability of quorum consensus protocols have been concerned with assessing the system availability. In this work we consider the user point-of-view availability defined as the probability that a mutual exclusion operation originating at a given site can proceed to completion. We consider scenarios where the network links, as well as the network nodes may fail. Our objective is to analyze the user experienced availability and to determine how to best design a system so as to obtain high availability.

1 Introduction

A distributed system consists of a number of cooperating nodes interconnected by a communication network. The nodes communicate with each other through messages sent over the network. The advent of high speed networking allows for the possibility of running a multitude of new distributed applications. A number of these applications require mutually exclusive access to resources, for example, updates to a file must be synchronized. Synchronization methods used in distributed systems must be tolerant to node and network failures.

A general class of protocols used for achieving mutual exclusion in distributed systems is *quorum consensus*. In these methods, an operation must obtain permission from a group of *coordinators*, before it can proceed to completion. The groups that can grant permission must intersect with each other and a coordinator grants permission to only one operation at a time. This ensures that no two operations can proceed simultaneously. A set of groups, known as a *coterie* [1, 2], can be defined whose members are groups of coordinators that have the non-empty intersection (i.e., contain at least one common coordinator) property. In addition, if a group is a member of the coterie then it cannot be a subset of any other group in the coterie. The best known quorum consensus protocol is majority consensus [3] where each group consists of a majority of the coordinators. *Weighted Voting* [4] is a simple technique that can be used to implement quorum consensus protocols, where each node is assigned a number of votes and an operation must obtain a majority of votes before it can proceed to completion.

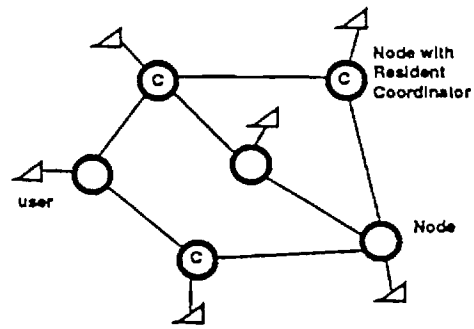


Figure 1: Coordinators, Users and Switching Nodes

Each assignment of votes uniquely defines a coterie. For example, in a system with four coordinators, A, B, C and D that are assigned 2, 3, 1, and 1 votes respectively, an operation requires at least four (a majority) votes to proceed. The coterie describing this is {{A,B}, {A,C,D}, {B,C}, {B,D}}. It has also been shown that there exist coteries that cannot be obtained from a vote assignment [2]. As will be demonstrated in this paper, non-vote assignable coteries may be needed to optimize system performance. Multi-dimensional voting is a voting-like technique that can be used to implement non-vote assignable coteries [5].

We consider a store-and-forward network with coordinators resident in some of the switching nodes. (See Figure 1.) The nodes and the links in the network are unreliable, and failure of a switching node where a coordinator resides implies that the coordinator is not accessible. Users are attached to the system by a connection to one of the switching nodes. A user becomes disconnected if the node to which he is attached fails.

The main motivation for having multiple coordinators is to enhance system availability. A system with a single

*This work was supported in part by NSF grants NCR-8604850 and CCR-8606358.

Paper # 19: Proceeding of INFOCOM 90
Submitted to Acta Informatica

Resource Finding in Store–and–Forward Networks*

José M. Bernabéu-Aubán *Mustaque Ahamad* *Mostafa H. Ammar*
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332

Abstract

We model the process of searching for a resource in a distributed system whose nodes are connected through a store–and–forward network. Based on this model, we show a lower bound on the number of messages needed for finding a resource when nothing is known about its location. The model also helps us establish some results about the complexity of finding optimal algorithms to locate a resource when the probability distribution for the location of the resource is known. We show that the optimization problem is NP-hard for general networks. Finally we develop an algorithm for tree networks which can be specialized to polynomial algorithms for a class of trees. (The polynomial algorithms can be used as the basis of heuristic algorithms for general networks.) An application of this algorithm for path networks can be adapted to find optimal search algorithms for bidirectional ring networks.

1 Introduction

Distributed systems need to implement algorithms for finding the location of remote resources to reduce the complexity of their use. We investigate the communication cost of location finding algorithms in a store–and–forward network. We consider two situations. In the first, our goal is to investigate resource finding algorithms when a searcher node does not know where information about the resource resides. In the second, we assume the searcher node has some statistical information (e.g., a probability distribution). Such situations can arise in a distributed system with the commonly used schemes such as name servers [1] or hint tables [2]. For example, when the name server node fails, the algorithm used by the node that wants to find the resource location (searcher node) must work without exact knowledge about the nodes that are likely to know the resource location.

When the searcher has no information about the location of a resource (or of its references), we show in this paper:
1) The expected number of messages used in searching for the resource has a lower bound of $O(\sqrt{\frac{\mu}{\lambda}N})$, where N is the number of nodes in the network, μ is the rate at which the resource moves and λ is the rate at which requests for the resource arrive. This lower bound includes the messages needed to update the references to the resource. Furthermore, there is a nondeterministic algorithm reaching this lower bound in a completely connected network. 2) For arbitrary networks, the expected number of messages needed to find the location of a resource that has n references in the network has an upper bound of $N - n$ (this bound is tight).

When the searcher has a probability distribution describing the likelihood of a particular node knowing the location of a resource, we show: 1) The problem of determining the optimal way of searching an arbitrary network to minimize the expected number of messages used is NP-hard. 2) For complete networks, in which the cost of sending a message through a link is the same for all links, the problem of selecting the optimal way to search the network is shown to be equivalent to sorting. 3) An algorithm is developed for tree networks which improves on exhaustive search. We show a polynomial algorithm to find the optimal way to search for a resource in a bidirectional ring network.

The resource finding problem has been addressed by several researchers. In [3] and [4], methods suitable for store–and–forward networks are presented and it is shown that their average cost when the ratio of resource request and movement rate is a constant, is $O(\sqrt{N})$ in complete networks. However, these methods require that additional information be used by each node. For example, two of the forwarding protocols studied in [5] require that all nodes store an address for each of the resources. Similarly, in [3], each node must have two sets of nodes associated with it.

The problem of searching has also been addressed in the literature in a different context [6,7]. However, the solutions obtained are not applicable to location finding in distributed systems. There exist other schemes which are useful in a particular type of network [8,9]. These schemes

*This work was supported in part by NSF grants CCR–8806358 and NCR–8604850.

Paper #10 : Proceedings of the 13th Conference on local Computer Networks

Using Multicast Communication to Locate Resources in a LAN-Based Distributed System†

Mustaque Ahamad *Mostafa H. Ammar* *José M. Bernabéu-Aubán* *M. Yousef Khalidi*

School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332

Abstract

In this paper we present a resource (e.g., file, process) location scheme which exploits the multicast communication capability of local area networks. In the scheme, the universe of resource names is partitioned into a relatively small number of groups and each group is assigned a unique address. Nodes storing the locations of resources belonging to a particular group instruct their network interfaces to receive all location messages sent to the group address. To locate a resource, a node first determines the address of the group to which the resource belongs (this can be accomplished via a well-known hash function), and a multicast message is then sent to the address. The algorithm performance is studied by means of simulation, and approximate closed form solutions are derived for systems operating at heavy and low loads. The scheme's performance is compared with that of broadcast, and it is shown that the proposed scheme performs much better than broadcast alone.

1 Introduction

The advantages offered by distributed systems include resource sharing, fault-tolerance and parallel execution of a computation. The programming of distributed systems is more complex than centralized ones due to the unavailability of the global state of the system. For example, in a dynamic system where resources (e.g., files, processes) can be migrated between nodes, a user must program an algorithm to find the current location of a resource needed by his or her computation. This can be avoided if users are provided with the abstraction of a unified system where the location of resources is transparent to them. Resources are referred to by names and, at runtime, the system determines the current location of a named resource.

Many schemes have been proposed for finding the location of a named resource. Conceptually, there exists a database that stores the associations between resource names and their locations. This database can be partitioned and stored at one or more nodes that are called *name servers*. When a remote resource, R, needs to be accessed, the request for its location should be sent to a name server that stores R's location. The system must also implement algorithms to update the information stored by the name servers when resources are created or deleted or when they are migrated. To avoid this, the database can be distributed in such a way that a name server at a node maintains a list of only resources local to the node. In such a system a remote resource can be located by broadcasting its name, and having the node where the resource is located respond. This scheme is used in Clouds [DLS85] for locating remote objects. Broadcast can also be used when other schemes fail to locate a resource.

We are concerned with a distributed system that uses a broadcast bus local area network. In such an environment, all network interfaces receive every message carried on the bus. A particular message is delivered to the attached node only if it is sent to a destination address that the interface has been instructed to recognize. Such addresses will at least include the broadcast address and the node's own address. Thus, if a broadcast message is used to locate a resource, the message will be delivered to all the nodes in the distributed system. This in turn will cause all the nodes to search their local resource directories which represents a wastage of CPU time at all nodes except the one where the resource resides.

In this paper, we explore the design of a distributed name server where multicast communication is used to locate the requested resource. In such a system, a particular message sent to locate a resource will be delivered to only a subset of the nodes in the system. The availability of bus interface communications technology that supports multicast in the hardware provides the motivation for this work. Our goal is to design a location scheme that is simple from the point of view of a node that needs to find a resource but, at the same time, reduces the number of nodes that must participate in the location process.

We associate a multicast address with each resource name and this address is used to communicate with the name server of the resource. Each node receives messages sent to multicast addresses corresponding to the resources whose locations are stored by the local name server. Typically, a limited number of multicast addresses will be available at each interface for use by the resource location operations. Since the number of resources in the distributed system can be large, the resource name to multicast address mapping is many-to-one. For such a system, we present the algorithms to be executed when a resource is created, deleted or a request is made for finding its location. We also study the performance of the multicast scheme and compare it with broadcast. The cost measure used is the number of nodes that process messages sent for finding a resource or for updat-

†This work has been partially supported by NSF grants CCR-8306358, NCR-8604850, and CCR-8619886.

Optimal Selection of Multicast Groups for Resource Location in a Distributed System[†]

José M. Bernabéu-Aubán *Mostafa H. Ammar* *Mustaque Ahamad*

School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332

Abstract

In this paper we present a protocol to locate (or find) named resources in a distributed system which uses the multicast capabilities of the underlying network. Each node in the network uses a sequence of node groups, and each node group is associated with a unique multicast address. To locate a resource, the searching node sequentially polls each one of the groups until the resource is found. This scheme is a generalization of both pure polling and broadcast. Our basic aim is to show how to obtain an optimal division of the nodes into multicast groups. To that end, the protocol is analyzed and an efficient algorithm is given that provides a group division minimizing the expected cost per location operation.

1 Introduction

Distributed systems offer many advantages over centralized ones, including fault-tolerance, resource sharing and increased parallelism. The task of programming a distributed system, however, is more difficult because the global system state is not available. In particular, in systems where resources (e.g., files, processes) can be migrated between nodes, it would be necessary for the user to implement a procedure to find resources needed by the computation. To avoid this, the system must offer the users the abstraction of a unified system in which the location of resources is transparent to them. In such a system, the Operating System should implement an algorithm to find the location of a remote resource. When a resource may be used repeatedly at a node, caching its address locally [1,2,3,4], is a widely used technique for reducing the cost of determining the location of a resource. Since the cache information may be incorrect, the general problem of finding the resource still exists when caching is used.

A widely used scheme to find resources involves the use of name servers [1,5]. In its simplest form, one of the nodes in the network is designated as the name server for the whole system. When a node needs to locate a resource, it directs a request to the name server. When a resource moves between nodes, an update message is sent to notify the name server. In a large system, such a name server would become a bottleneck, degrading the performance of the system. Also, the single name server approach would be especially vulnerable to node failures. A more general approach can distribute the name server task among several nodes and, a particular name server usually takes care of only a part of the resource name space. The problem now is to decide which name server to contact to find the location of a resource. A resource's location may now be found by broadcasting (actually multicasting [6,7]) to all name servers requesting that they provide the resource's address. Another approach, used in the R* system [2], is to encode the name of the node where a resource was created in the resource's name. Then that node will function as the resource's name server.

In the absence of name servers, a node wishing to determine the location of a resource, can send a broadcast message to all nodes and make them search their local directories. This is the approach taken in the *Clouds* operating system [8]. Broadcasting, though simple, would waste computational resources at every node, where it would compete with the local computations for CPU time. For large rates of location requests this would rapidly degrade the performance of the entire system.

At the other extreme, if the individual nodes are polled sequentially, this would certainly decrease the amount of CPU time wasted in the system (especially if the nodes more likely to know about the resource were consulted first). However this approach would also increase the bandwidth utilization, because many messages will be sent. Since the messages are sent sequentially, the real disadvantage of this approach is that the location operations would take longer (larger response time).

In this paper we present a location protocol which considers a cost measure that includes both the CPU utilization and the response time. The approach taken is based on a scheme in which the nodes in the network are divided into disjoint multicast groups, and are polled by a sequence of multicast messages. The two approaches mentioned above are just special cases when all nodes are reached by a single message (broadcast) or when only one node is reached with each message (polling). We also present a cost model for the system and an efficient algorithm which, based on the probability distribution of a resource's location among the nodes in the network, finds the optimal decomposition into disjoint groups, as well as the optimal sequence in which the groups should be polled.

In section 2 we give a description of the protocol operation. In section 3 we present the model of the system to be used for the cost analysis carried out in section 4. Section 5 describes an algorithm to determine an optimal multicast grouping. Some numerical examples are presented in section 6. In section 7 we

[†]This work was supported in part by NSF grants NCR-8604850 and CCR-8506358.

Multi-Dimensional Voting: A General Method for Implementing Synchronization in Distributed Systems*

Shun Yan Cheung *Mustaque Ahamad* *Mostafa H. Ammar*
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

Abstract

We introduce a new concept, *multi-dimensional voting*, in which the vote and quorum assignments are k-dimensional vectors of non-negative integers and each dimension is independent of the others. Multi-dimensional voting is more powerful than traditional weighted voting because it is equivalent to the general method for achieving synchronization in distributed systems which is based on coteries (set of groups of nodes) but its implementation is easier than coteries. We describe an efficient algorithm for finding a multi-dimensional vote assignment for any given coterie and show examples of its use. We also show how multi-dimensional voting can be used to easily implement novel algorithms for synchronizing access to replicated data or to ensure mutual exclusion. These algorithm cannot be implemented by traditional weighted voting.

1 Introduction

Distributed systems offer many advantages, including resource sharing and fault-tolerance. The latter can be achieved by replicating a resource at nodes with independent failure modes. Replication can also improve performance when load is shared among the nodes that have instances of a resource. In many applications, users need to synchronize access to shared resources. For example, when data is replicated to improve its availability, updating the file requires mutually exclusive access. This is necessary for maintaining the consistency of the data. The synchronization technique should work in the presence of node and communication failures.

An operation that requires mutual exclusion can be executed if permission can be obtained from a group of nodes. In general, a node can execute the operation if permission can be obtained from any one group in a set of intersecting groups [1]. Such a set is called a *coterie* in [2]. For reading and writing of replicated data when several readers are allowed to access the data concurrently, read and write coteries can be defined in a similar way [3]. Another well-known synchronization method is weighted voting [4] which is a generalization of the majority consensus method [5]. In voting, each node is assigned a number of votes and each operation must obtain a pre-defined quorum of votes before it is allowed to execute to completion. Voting can be used for achieving mutual exclusion and synchronizing reading and writing of replicated data. In mutual exclusion, each operation must obtain a majority of the votes assigned before it can proceed. In reading and writing, the read and write quorums must be such that their sum is more than the total number of votes and the write quorum is at least a majority of all votes.

Voting is appealing because it is flexible and can be easily implemented. Each node in voting stores its assigned vote and when it wants to execute an operation that requires q votes, it communicates with other nodes to request their votes. The execution of the operation can proceed if the sum of the votes received is at least q. In contrast, in a system that uses coteries, operations must know all the groups of the coterie and test if the nodes that responded positively to its request form a group of the coterie. Voting is also more flexible. Adding or removing a node requires only a change of the quorum and assigning the proper number of votes to the new node. In a coterie-based system, adding and removing a node may cause the addition and deletion of numerous groups. However, Garcia-Molina and Barbara proved in [2] that the method of coteries is more general than voting by showing coteries which cannot be obtained from any vote assignment. Coteries that are not obtained from vote assignments can be used to achieve better performance by reducing the number of messages. For example, structured coteries as those used in the methods described in [6, 7] have lower communication cost and they cannot be implemented by voting.

We present in this work a new voting based method that is as powerful as the method of coteries and has the flexibility and ease of implementation of voting. In multi-dimensional voting, the vote assignment to each node and the quorums are k-dimensional vectors of non-negative integers. Each dimension of the vote and quorum assignment is similar to voting and the quorum requirements in different dimensions can be combined in a number of ways.

*This work was supported in part by NSF grants NCR-8604850 and CCR-8806358.

Paper #13

Submitted for Publication to: INFORMATION PROCESSING LETTERS

On the Optimality of Voting*

Shun Yan Cheung
Mostafa H. Ammar
Mustaque Ahamad

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332

Abstract

We study the problem of optimizing availability of a distributed system for mutual exclusion *and* reading and writing of replicated data when the synchronization schemes used are based on consensus of groups of nodes. We show that the set of groups (or quorum set) defined by a *best-behaved* vote assignment will provide the highest availability. An algorithm for finding a best-behaved vote assignment for a given distributed system is also presented.

Keywords: concurrency, databases, distributed systems, fault tolerance, performance evaluation.

*This work was supported in part by NSF grants NCR-8604850 and CCR-8806358.

Paper #14

Prototyping a Broadcast Delivery Information System

Mostafa H. Ammar
Hyoung-Joo Kim
Georgia Institute of Technology
GIT-ICS-90/16

May 3, 1990

Abstract

One of the challenges faced by an information system designer is how to configure a low cost system that can support a potentially large user population and still provide good response time. In typical systems the response time increases quickly with load. In this report we consider how to improve the response time performance of an information delivery system by exploiting the inevitable commonality of information need present in a large user base. We use the broadcast delivery of responses in conjunction with a design that allows the response to a user's request to satisfy other requests. We present a preliminary design for a prototype Broadcast Delivery Information System that allows general interactive database access without requiring expensive non-standard hardware. We also discuss how such a system may be configured using equipment already available in the School's Telecommunications Laboratory.

Authors' Address: School of Information and Computer Science Georgia Institute of Technology, Atlanta, GA 30332

Comments on the contents of this preliminary design document are solicited.

Paper # 13

Scheduling Algorithms for Broadcast Delivery Information Systems

*H. D. Dykeman**
*J. W. Wong**
*M. H. Ammar***

Technical Report GIT-ICS-88/35

* Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

** School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

October 17, 1988

Abstract

A system which uses a broadcast channel to deliver information to a community of users is considered. Information is organized into units called pages. Requests for information pages are processed by a service computer, and the requested pages are broadcast to all users. The use of broadcast delivery is attractive from the viewpoint of response time performance because a single transmission of a page will satisfy all pending requests for that page. An important design issue is to determine the scheduling algorithm which selects the next page to be broadcast. This scheduling problem is formulated as a Markov decision process, to identify the properties of a good scheduling algorithm. Using these properties, three new scheduling algorithms are proposed, and their performance is evaluated using simulation. Implementation issues for the proposed algorithms are also discussed.

Paper #16 : Submitted to IEEE Computer Magazine

Replicated Data Management in Distributed Systems*

Mustaque Ahamad
Mostafa H. Ammar
College of Computing
Georgia Institute of Technology,
Atlanta, GA 30332
mustaq@cc.gatech.edu, (404) 894-2593
ammar@cc.gatech.edu, (404) 894-3292

Shun Yan Cheung
Department of Mathematics and Computer Science
Emory University,
Atlanta, GA 30322
cheung@mathcs.emory.edu, (404) 727-3823

*This work was supported in part by NSF grants NCR-8604850 and CCR-8806358.

ABSTRACT

Replication of data in a distributed system is a way to enhance the performance of applications that access the data. A system where data is replicated can provide better fault tolerance capabilities as well as improved response time. However, such improvement is achieved at the expense of having to manage replication by implementing replica control protocols. Such protocols are required to insure that data consistency is maintained in the face of system failures. In this article we describe the issues involved in maintaining the consistency of a replicated database system. We next describe three basic techniques for managing replicated data and discuss the relative merits of each technique. This is followed by a survey of extensions to the basic approaches. A discussion of future directions in research on data replication concludes our presentation.

Key Words: Distributed Systems, Data Replication, Fault Tolerance, Replica Control Protocols.

Selected Publications

Paper #3

Optimizing Vote and Quorum Assignments for Reading and Writing Replicated Data

Shun Yan Cheung
Mustaque Ahamad
Mostafa H. Ammar

Reprinted from
IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING
Vol. 1, No. 3, September 1989

Optimizing Vote and Quorum Assignments for Reading and Writing Replicated Data

SHUN YAN CHEUNG, MUSTAQUE AHAMAD, AND MOSTAFA H. AMMAR, MEMBER, IEEE

Abstract—**In the weighted voting protocol which is used to maintain the consistency of replicated data, the availability of the data to read and write operations not only depends on the availability of the nodes storing the data but also on the vote and quorum assignments used. We consider the problem of determining the vote and quorum assignments that yield the best performance in a distributed system where node availabilities can be different and the mix of the read and write operations is arbitrary. The optimal vote and quorum assignments depend not only on the system parameters such as node availability and operation mix, but also on the performance measure. We present an enumeration algorithm that can be used to find the vote and quorum assignments that need to be considered for achieving optimal performance. When the performance measure is data availability, an analytical method is derived to evaluate it for any vote and quorum assignment. This method and the enumeration algorithm are used to find the optimal vote and quorum assignment for several systems. The enumeration algorithm can also be used to obtain the optimal performance when other measures are considered.**

Index Terms—**Availability, data replication, fault tolerance, replica control methods, vote and quorum assignment, weighted voting.**

I. Introduction

A DISTRIBUTED system consists of a number of potentially unreliable nodes interconnected via a communication subnetwork. The resources stored at the nodes can be shared and when a node fails, the resources stored at the node become unavailable. Replicating resources at different nodes with independent failure modes can enhance availability and fault tolerance, since a resource could be available even when some nodes have failed. When data are replicated, care must be taken to preserve consistency among the various copies or *replicas*. In addition to increased availability, replication can also provide improved performance of read transactions by reducing the network communication cost since these transactions can access the data from the local replica.

A large number of replica control protocols have been developed to maintain the consistency of replicated data [1]. In this paper, we address the issue of optimization for a voting-based replica control protocol by deriving a general method for finding the optimal settings for the parameters of the protocol. We consider the voting mechanism because it has proven to be flexible and relatively easy to implement.

Voting has been used for various applications in distributed systems. In [2], Gifford proposed its use for synchronizing read and write operations on replicated files. Each file replica is assigned some number of votes and each operation is required to obtain a predefined quorum of votes to proceed. To ensure that a read operation returns the value installed by the last write operation, the read and write operations must acquire r and w number of votes, respectively, such that $r + w > L$, where L is the total number of votes assigned to all replicas. The values r and w are called the read and write quorum. Generally, $r + w = L + 1$ is used which ensures that each read quorum has a nonempty intersection with each write quorum. Since all replicas need not be updated when a write operation completes, timestamps or version numbers must be used in order to determine the value that is written most recently. When version numbers are used, each write quorum must also intersect with every other write quorum, i.e., $2w > L$ [2].

A number of replica control protocols have been derived from weighted voting. Eager and Sevcik introduced a dynamic scheme based on voting that allows the system to switch between normal and failure modes [3] (which have different values for read and write quorums). The system can also change the quorum assignment in the schemes presented in [4]–[6] and the vote assignment can be changed in the scheme described in [7]. Other protocols based on voting are presented in [8]–[10].

The problem of assigning votes to achieve mutual exclusion is addressed by Garcia-Molina and Barbara in [11]. When the quorum for each operation is a majority of all votes assigned, each operation will have mutually exclusive access to the data. In general, mutual exclusion can be guaranteed by defining a set of groups of nodes [12], called a coterie, such that any two groups in a coterie have a nonempty intersection. When voting is used, the groups of nodes that have a majority of the votes constitute a coterie (there exist coteries that cannot be obtained from any vote assignment [11]). In [11], it is shown that only a finite set of vote assignments need to be considered to get all coteries that can be obtained from vote assignments. Thus, it is not necessary to deal with the unbounded set of possible vote assignments. In another work, the same authors have considered the problem of

Manuscript received September 28, 1988; revised July 11, 1989. This work was supported in part by the National Science Foundation under Grants CCR-8806358 and NCR-8604850.

The authors are with the School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA 30332.

IEEE Log Number 8930638.

selecting the vote assignment that results in the highest system availability for mutual exclusion [13]. For a system where all node availabilities are the same, they derived the optimal vote assignment. In [14], Tong and Kain presented an algorithm for assigning votes that maximizes system availability for mutual exclusion in a system where the availability of the nodes can be different. In a related work [15], the authors evaluate the use of the voting mechanism to manage read and write transactions. For systems where node availabilities are identical, values are derived for the optimal degree of replication and for the optimal read quorum.

We consider the problem of optimizing performance for reading and writing replicated data. Since a direct method such as the one described in [14] does not exist for optimizing performance for reading and writing, our approach first identifies the set of vote assignments that need to be considered and then chooses the optimal one. To this end, we present the following techniques which are the major contribution of this paper.

1) A Technique to Enumerate the Vote and Quorum Assignments That Need to be Considered to Optimize a Given Performance Measure for the General Read/Write Case: The measure of interest may be data availability, communication cost, response time, or a combination of these measures. At the heart of our approach is an efficient algorithm to generate the vote and quorum assignments that need to be considered in the optimization. Such enumeration algorithms have only been considered in the literature for the special case of majority quorums. This paper provides a method to enumerate vote and quorum assignments in the general read/write case when the read and write quorums may be different. Quorums (other than the majority) may be useful in improving performance by reducing the cost of more frequent operations.

2) A Technique for Evaluating Availability: We use system availability to illustrate how the set of vote and quorum assignments can be used to find the optimal assignment for reading and writing replicated data. We present an efficient algorithm to evaluate the availability for a system with different node availabilities and arbitrary vote and quorum assignments. Previous methods for evaluating availability are based on set inclusion/exclusion [16] which cannot be used efficiently for larger systems where data are replicated at over 20 nodes.

The paper is organized as follows. We describe the enumeration algorithm in Section II. Section III presents a model of the system and the analysis that derives a method for finding the availability. Some numerical examples are presented in Section IV and we conclude the paper in Section V.

II. Enumerating Vote Assignable Read Coteries

A. Definitions

Let N be the number of nodes that store replicas of a data item. A data item may correspond to a file or several items may be stored in one file. We number the nodes that store the data from 1 to N. Replica i resides at node i and let $U_N = \{1, 2, \cdots, N\}$ be the universe of the replicas. There are two types of operations allowed on the replicas, read and write, and each operation must acquire the consensus of a number of replicas to proceed. A *read group* G is a subset of U_N and is a minimal group of replicas such that a read operation can proceed if all replicas in the group are available (i.e., the nodes where the replicas are stored have not failed). Thus, failure to acquire the consensus of all replicas in any read group causes the read operation to block or abort. A *read coterie* Q_r is a collection of read groups satisfying the following *noncontainment* property, for any read groups G and H:

$$\forall G, H \in Q_r: G \not\subset H.$$

The noncontainment property is a result of the fact that each group is minimal. For instance, if $G \subset H$, then H is not minimal because even when $i \in H - G$ is removed from it, a read operation can still be completed (all replicas in G are available).

A write operation can proceed only if it can acquire the consensus of replicas that constitute a write group. The *write coterie* Q_w corresponding to a given read coterie Q_r is unique and consists of write groups H that satisfy the noncontainment property and also for each $H \in Q_w$:

$$\forall G \in Q_r: G \cap H \neq \phi.$$

We assume in this paper that timestamping is used to identify the current value. When version numbers are used for this purpose, the intersection of any two write groups must also be nonempty. The results of this paper can be modified to accommodate this case.

In voting [2], a special subset of read/write coteries is used, namely those that can be obtained from some vote assignment, and we will call these read/write coteries *vote assignable*. A vote assignment is a vector $V_N = (v_1, v_2, \cdots, v_N)$ where v_i $(1 \leq i \leq N)$ is a nonnegative integer representing the number of votes assigned to replica i. We define $L(V_N)$ to be the total number of votes assigned to the replicas, or $L(V_N) = \Sigma_{i=1}^{N} v_i$. Let group $G = \{g_1, g_2, \cdots, g_k\}$ be a set of replicas where replica g_1 has the least number of votes and we denote by $v(G)$ the sum of the votes assigned to replicas in G. The group G is a *tight group* of s number of votes if and only if the replicas in G collectively have at least s votes and removal of any replica (in particular the replica g_1 which has the least number of votes) from G leaves a group with less than s votes. In other words,

$$G \text{ is a tight group of } s \text{ votes}$$
$$\Leftrightarrow s \leq v(G) \leq (s - 1) + v_{g1}.$$

A *read* group of r votes is a tight group of r votes. The value r is called the read quorum. A *read coterie* with quorum r consists of all the read groups of r votes. A vote assignment V_N and a read quorum r uniquely identify a read coterie. However, the same read coterie can be obtained using different vote assignments and/or different read quorums. For example, the read coterie $\{\{1, 2\}, \{3\}\}$ can be obtained from $V_1 = (1, 1, 2)$ and $r = 2$ and

$V_2 = (2, 3, 5)$ and $r = 4$. For each read coterie with a quorum of r votes there exists a write coterie with a quorum of $L + 1 - r$ votes. The write coterie for a given read coterie is unique and we can limit our attention in the enumeration algorithm to only the read coteries.

For a system with N nodes, we use $Q_N(r, V_N)$ to denote the read coterie obtained from the vote assignment V_N when the read quorum is r. For $1 \leq r \leq L(V_N)$, the read coterie is well-defined. If $r > L(V_N)$, then we define $Q_N(r, V_N) = \phi$, i.e.; no read group can be formed. For $r \leq 0$, we define $Q_N(r, V_N)$ to be $\{\phi\}$, i.e., a read operation requires no consensus.

A vote assignable read coterie of N nodes corresponds to a vote and quorum assignment V_N and r, respectively. To determine the optimal assignment, we only need to consider the set of (V_N, r) pairs such that they represent all distinct vote assignable read coteries. We denote this universe of read coteries for N replicas by Ω_N.

Two read coteries R and S are *isomorphic* if and only if there is a permutation π of integers $1, \cdots, N$ such that when we replace each i in S by $\pi(i)$, we obtain R. If the vote assignment V_N is permuted and the read quorum r is kept at the same value, the resulting read coterie will be isomorphic to $Q_N(r, V_N)$. (Permuting the vote assignment will, in general, affect the performance in a system with different node availabilities.) A collection of read coteries E_N is an *enumeration* [11] if every read coterie in Ω_N is either in E_N or is isomorphic to one in E_N and no two read coteries in E_N are isomorphic. Note that $E_N \subseteq \Omega_N$ and E_N can be obtained from Ω_N by choosing one representative from each isomorphic class. Conversely, Ω_N can be obtained from E_N by applying all possible permutations of $1, 2, \cdots, N$ to the members of E_N. In general, Ω_N is the space that must be searched to find the best vote assignable read coterie for a given performance measure.

A summary of the terms used and the notation is presented in Table I.

TABLE I
SUMMARY OF TERMINOLOGY AND NOTATION

Terminology	Notation	Description
Universe of Replicas	U_N	Set of all N replicas
Vote assignment	$\underline{V}_N$	v_i = number of votes assigned to node i
Total Number of Votes	$L(\underline{V}_N)$	$\sum_{i=1}^{N} v_i$
Tight group of s votes		Group that has at least s number of votes and removal of any member leaves a group with less than s votes
Read/write quorum	r/w	Threshold of votes for reading/writing $(r + w = L(\underline{V}_N) + 1)$
Read group of r votes	$G, H, \ldots$	Tight group of r votes
Read Coterie	$Q_N(r, \underline{V}_N)$	Set of all read groups of r votes
Write group of w votes	$G, H, \ldots$	Tight group of w votes
Write Coterie	$Q_N(w, \underline{V}_N)$	Set of all write groups of w votes
Universe of Read Coteries	Ω_N	Set of all read coteries of N copies
Enumeration of Read Coteries	E_N	Set of all non-isomorphic read coteries of N copies

B. Generating All Vote Assignable Read Coteries

We now present an algorithm that can be used to generate Ω_{N+1} when Ω_N is given. Since $\Omega_1 = \{\{\phi\}, \{\{1\}\}, \phi\}$, the algorithm can be used, in principle, to find Ω_N, for any value of N. The algorithm is derived from the results of the following lemma that states how the read coterie of a system with N replicas changes when the system is expanded by creating a new replica (replica $N + 1$) which is assigned v_{N+1} votes.

Lemma 2.1—Expanding the Vote Assignment: Let $V_N = (v_1, v_2, \cdots, v_N)$ be a vote assignment to a system of N replicas and let V_{N+1} be the vote assignment that results when replica $N + 1$ having v_{N+1} votes is added to the system (replicas 1 to N are assigned the same number of votes in both V_N and V_{N+1}). Then,

$$Q_{N+1}(r, V_{N+1}) = Q_N(r, V_N) \cup \{G \cup \{N+1\} \mid G \in G_N(r - v_{N+1}, V_N) \wedge G \notin Q_N(r, V_N)\}.$$

Proof: See Appendix A.

We know from Lemma 2.1 that for each vote assignable read coterie $Q_{N+1}(r, V_{N+1})$ of $N + 1$ replicas there must exist read coteries Q_1 and Q_2 of N replicas ($Q_1 = Q_N(r, V_N)$ and $Q_2 = Q_N(r - v_{N+1}, V_N)$) such that $Q_{N+1}(r, V_{N+1})$ is related to Q_1 and Q_2 as stated in the lemma. The algorithm presented in Fig. 1 uses this fact as it generates read coteries of a system of $N + 1$ replicas by combining every pair of the read coteries of N replicas using the relationship defined by Lemma 2.1 Notice that since Q_1 and Q_2 in Lemma 2.1 have the same vote assignment and we are combining all pairs (even those which are obtained from different vote assignments), the algorithm has to check that the resulting set can be obtained by some vote assignment. The output of the algorithm, Ω, will be Ω_{N+1}. This follows from the fact that each read coterie $Q \in \Omega_{N+1}$ can be written as a combination of some pair of read coteries in Ω_N (as given by Lemma 2.1) and every possible pair of read coteries of Ω_N is combined by the algorithm in the manner given by Lemma 2.1, hence, Q will be generated.

Note that Q, the set generated by the statement (†) of the algorithm, may not satisfy the noncontainment property for read coteries. For example, $Q_1 = \{\{1\}\}$, $Q_2 = \{\{1, 2\}\}$ and $N + 1 = 3$, the set $Q = \{\{1\}, \{1, 2, 3\}\}$ is generated which violates the noncontainment property. However, such sets are not vote assignable and will not be included in Ω. The statement (†) also generates read coteries (satisfying the noncontainment property) that are not vote assignable.

Determining whether the set Q is vote assignable can be done by formulating a linear program (LP) from the groups of Q. We define the following set of groups of $N + 1$ replicas:

$$Y(Q) = \{H \in 2^{U_{N+1}} \mid H \text{ is not a superset of any group of } Q \text{ or } H \text{ is a proper subset of a group of } Q\}$$

where $2^{U_{N+1}}$ is the set of all subsets of the universe of $N + 1$ nodes. When a group G is in Q, any proper subset of

```
Ω := φ; (Ω is the output of the algorithm)
for every Q1 ∈ ΩN do
    for every Q2 ∈ ΩN do
        Q := Q1 ∪ {G ∪ {N + 1} | G ∈ Q2 ∧ G ∉ Q1}; ....(†)
        if Q is obtainable from some (V_{N+1}, r) and Q ∉ Ω then
            Record (V_{N+1}, r);
            Ω := Ω ∪ Q;
```

Fig. 1. Algorithm 1.

G must have less than r votes. Also when a group H does not contain any group of Q, it cannot have r or more votes (if it did, then H or its subset must be in Q). The set $Y(Q)$ is thus the set of all groups that do not have the required quorum of r votes. We use this property of $Y(Q)$, coupled with the fact that all the groups in Q must have at least r votes, to formulate the following LP:

$$\min \sum_{i=1}^{N+1} v_i + r$$

$$\text{s.t.: } \forall G \in Q: \sum_{g \in G} v_g \geq r$$

$$\forall H \in Y(Q): \sum_{h \in H} v_h \leq r - 1$$

$$v_i \geq 0,\ i = 1, 2, \cdots, N+1$$

$$r \geq 1.$$

If the LP does not have a solution, then there are no values V_{N+1} and r that satisfy the constraints. If the LP does produce a solution, the values of V_{N+1} and r are rational and since multiplying V_{N+1} and r by the same value will not affect the resulting read coterie, we can always convert the solution to an integer vote assignment and quorum. In principle, we are only concerned with obtaining a feasible solution to the LP. However, the complexity of the performance analysis method described in Section III is a function of the votes v_i and quorum r. We are thus using the indicated objective function.

C. Generating An Enumeration of Vote Assignable Read Coteries

The complexity of Algorithm 1 depends on the size of the input set Ω_N. Also note that, for a given N, we only need to generate the enumeration set E_N from which Ω_N can be obtained. Due to the fact that, in general, E_N is much smaller than Ω_N (e.g., $|E_5| = 119$ and $|\Omega_5| = 3287$), an algorithm which generates E_{N+1} when E_N is given as input, will require less CPU time. Such an algorithm can be derived from Algorithm 1 by only including a single member from a class of isomorphic read coteries. The following lemma provides a simple technique to achieve this and the resulting enumeration algorithm is shown in Fig. 2.

Lemma 2.2—Equality of Isomorphic Read Coteries of Nondecreasing Vote Assignments: Let $V_N = (v_1, v_2, \cdots, v_N)$ and $W_N = (w_1, w_2, \cdots, w_N)$ be two nondecreasing vote assignments, i.e., $v_1 \leq v_2 \leq \cdots \leq v_N$ and $w_1 \leq w_2 \leq \cdots \leq w_N$. $Q_N(r, V_N)$ and $Q_N(s, W_N)$ are isomorphic if and only if $Q_N(r, V_N) = Q_N(s, W_N)$.

```
E := φ; (E is the output of the algorithm)
for every Q1 ∈ EN do
    for every Q2 ∈ EN do
        Q := Q1 ∪ {G ∪ {N + 1} | G ∈ Q2 ∧ G ∉ Q1};
        if Q is obtainable from some (V_{N+1}, r) then
            V'_{N+1} := sort(V_{N+1}); (sort in non-decreasing order)
            if Q_{N+1}(r, V'_{N+1}) ∉ E then
                Record (V'_{N+1}, r);
                E := E ∪ Q_{N+1}(r, V'_{N+1});
```

Fig. 2. Algorithm 2.

Proof: See Appendix A.

The above lemma shows that in each class of isomorphic read coteries, there is *exactly one* member that is obtained from a nondecreasing vote assignment, i.e., two read coteries that are obtained from nondecreasing vote assignments are either equal or nonisomorphic. Then, sorting the vote vector in nondecreasing order in Algorithm 2 will guarantee that $E = E_{N+1}$. The following theorem shows the correctness of Algorithm 2.

Theorem 2.1—Enumeration of Vote Assignable Read Coteries: Let E be the output generated by Algorithm 2, then $E = E_{N+1}$ which is the enumeration of Ω_{N+1} that contains only read coteries that have nondecreasing vote assignments.

Proof:

Claim 1: Every vote assignable read coterie of $N + 1$ replicas is either in E or is isomorphic to a read coterie in E.

Let $Q_{N+1}(r, V_{N+1})$ be an arbitrary read coterie of $N + 1$ replicas and V'_{N+1} be the nondecreasing vote assignment obtained from V_{N+1}. $Q_{N+1}(r, V'_{N+1})$ and $Q_{N+1}(r, V_{N+1})$ are isomorphic and by Lemma 2.1 we can write $Q_{N+1}(r, V'_{N+1})$ as

$$Q_{N+1}(r, V'_{N+1}) = Q_N(r, V'_N) \cup \left\{ G \cup \{N+1\} \mid G \in Q_N(r - v'_{N+1}, V'_N) \wedge G \notin Q_N(r, V'_N) \right\}$$

and since $Q_N(r, V'_N)$ and $Q_N(r - v'_{N+1}, V'_N)$ are in E_N, $Q_{N+1}(r, V'_{N+1})$ is included in E.

Claim 2: There are no isomorphic read coteries in E.

Each vote assignment is sorted in nondecreasing order and by Lemma 2.2 we conclude that no two read coteries in E are isomorphic. □

Table II contains the vote assignments, quorums, and read coteries of E_4 which is produced by repeating the enumeration algorithm three times starting with $E_1 = \{\{\phi\}, \{\{1\}\}, \phi\}$. The read coteries $\{\phi\}$ and ϕ are not included in the table but are elements of E_4. We have also generated E_5, E_6, and E_7. These enumerations have 119, 1113, and 29375 members, respectively.[1]

[1]The enumeration method has so far produced only integral valued vote assignments for every LP generated. It is known that when the constraint matrix of the LP is totally unimodular, the solution is integral valued [17]. The constraint matrices generated by the algorithm do not satisfy this property.

TABLE II
READ COTERIES OF FOUR NODES

Index	Read Set	Vote assignment				
		v_1	v_2	v_3	v_4	r
1	{{4}}	0	0	0	1	1
2	{{3}, {4}}	0	0	1	1	1
3	{{34}}	0	0	1	1	2
4	{{2}, {3}, {4}}	0	1	1	1	1
5	{{23}, {24}, {34}}	0	1	1	1	2
6	{{234}}	0	1	1	1	3
7	{{1}, {23}}	0	1	1	2	2
8	{{24}, {34}}	0	1	1	2	3
9	{{3}, {2}, {4}, {1}}	1	1	1	1	1
10	{{12}, {13}, {14}, {23}, {24}, {34}}	1	1	1	1	2
11	{{123}, {124}, {134}, {234}}	1	1	1	1	3
12	{{1234}}	1	1	1	1	4
13	{{4}, {12}, {13}, {23}}	1	1	1	2	2
14	{{14}, {24}, {34}, {123}}	1	1	1	2	3
15	{{124}, {134}, {234}}	1	1	1	2	4
16	{{4}, {123}}	1	1	1	3	3
17	{{14}, {24}, {34}}	1	1	1	3	4
18	{{3}, {1}, {12}}	1	1	2	2	2
19	{{13}, {14}, {23}, {24}, {34}}	1	1	2	2	3
20	{{34}, {123}, {124}}	1	1	2	2	4
21	{{34}, {234}}	1	1	2	2	5
22	{{3}, {13}, {23}}	1	1	2	3	3
23	{{34}, {124}}	1	1	2	3	5
24	{{14}, {23}, {24}, {34}}	1	2	2	3	4
25	{{24}, {34}, {123}}	1	2	2	3	5

For a given N, since Ω_N can be obtained from E_N, optimal vote and quorum assignments can be determined for any particular performance measure by an exhaustive search. In what follows we illustrate this by considering the optimization of the availability of replicated data for read and write operations. Although there is a direct method for obtaining the settings for voting that maximizes the system availability for mutual exclusion [14], no direct method is known for finding optimal settings for reading and writing.

III. PERFORMANCE ANALYSIS OF WEIGHTED VOTING

We now develop a method for evaluating the availability of replicated data for a given vote assignment and read quorum when the system consists of N nodes and each node stores a replica of the data item. This method will be used to evaluate the performance of the vote and quorum assignments enumerated in the previous section in order to determine the best one.

For each node i, $i = 1, \cdots, N$, we assume that the mean time-to-fail is $1/\lambda_i$ and the mean time-to-repair is $1/\mu_i$. We also define the parameter p_i which represents the proportion of time the node is operational. We thus have

$$p_i = \frac{\frac{1}{\lambda_i}}{\frac{1}{\lambda_i} + \frac{1}{\mu_i}} = \frac{\mu_i}{\lambda_i + \mu_i}. \tag{1}$$

The parameter p_i will be called the *availability* of node i and can be viewed as the steady-state probability that the node is operational.[2] We will use the *availability vector* $\boldsymbol{P}$ to denote $(p_1, \cdots, p_N)$. The vote assignment under consideration is denoted by $\boldsymbol{V} = (v_1, \cdots, v_N)$ and we define $L = \sum_{i=1}^{N} v_i$ (we drop the subscript N from our earlier notation for simplicity).

The availability for operations requiring s votes is the proportion of time (or steady-state probability) that the total number of votes from all operational nodes is equal to or exceeds s. We denote this probability for given quorum s, availability vector $\boldsymbol{P}$, and vote assignment V by $\alpha_s(\boldsymbol{P}, \boldsymbol{V})$ ($s = r$ for read access, $s = w$ for write access, and $r + w = L + 1$). We use the system availability α as the performance measure in the analysis. The system availability for read/write transactions is equal to $\alpha(\boldsymbol{P}, \boldsymbol{V}) = f\alpha_r(\boldsymbol{P}, \boldsymbol{V}) + (1 - f)\alpha_w(\boldsymbol{P}, \boldsymbol{V})$, with f being the fraction of read operations.

In order to derive an expression for $\alpha_s(\boldsymbol{P}, \boldsymbol{V})$ for given values of s, $\boldsymbol{P}$, and $\boldsymbol{V}$, we define the state of the system $\boldsymbol{n} = (n_1, \cdots, n_N)$ where $n_i = 1$ if node i is operational and $n_i = 0$ otherwise. Let $P(\boldsymbol{n})$ be the steady-state probability that the system is in state $\boldsymbol{n}$. Observe that $P(\boldsymbol{n})$ is the probability that all nodes with $n_i = 1$ are operational and that all nodes with $n_i = 0$ have failed. Thus, we have

$$P(\boldsymbol{n}) = \prod_{i=1}^{N} (1 - p_i)^{(1-n_i)} p_i^{n_i} = \left[\prod_{i=1}^{N} (1 - p_i)\right] \prod_{i=1}^{N} \left(\frac{p_i}{1 - p_i}\right)^{n_i}. \tag{2}$$

We next obtain an expression for $Q(m)$, the steady-state probability that the total number of votes available in the system is m. This is given by the sum of all state probabilities in (2) such that $C(\boldsymbol{n}, \boldsymbol{V}) = \sum_{i=1}^{N} n_i v_i = m$. Thus, we get for $m = 0, 1, 2, \cdots, L$

$$Q(m) = \sum_{\text{all } \boldsymbol{n} \text{ s.t. } C(\boldsymbol{n}, \boldsymbol{V}) = m} P(\boldsymbol{n}) = \frac{1}{K} h(\boldsymbol{V}, m) \tag{3}$$

where we define

$$\frac{1}{K} = \prod_{i=1}^{N} (1 - p_i) \tag{4}$$

$$h(\boldsymbol{V}, m) = \sum_{\text{all } \boldsymbol{n} \text{ s.t. } C(\boldsymbol{n}, \boldsymbol{V}) = m} \prod_{i=1}^{N} q_i^{n_i} \tag{5}$$

and $q_i = p_i/(1 - p_i)$. Since the operation requires s votes, we obtain that

$$\alpha_s(\boldsymbol{P}, \boldsymbol{V}) = \sum_{m=s}^{L} Q(m) = \frac{1}{K} \sum_{m=s}^{L} h(\boldsymbol{V}, m). \tag{6}$$

[2] An operation can proceed to completion if it can find a required quorum within a certain time-out period. There may be several reasons why nodes are unable to reply within the given time-out period. A node can suffer from a hardware failure or it may be operational but isolated from the other nodes because of network failures. Also, long delays due to network congestion and slow response times due to overload can cause an operation to abort because of time-out. A node that is unable to respond within the time-out period is said to have failed.

Computing $h(V, m)$ can be accomplished by observing that[3]

$$h(V, m) = \sum_{\text{all } \boldsymbol{n} \text{ s.t. } C(\boldsymbol{n}, V) = m, n_N = 0} \prod_{i=1}^{N} q_i^{n_i} + \sum_{\text{all } \boldsymbol{n} \text{ s.t. } C(\boldsymbol{n}, V) = m, n_N = 1} \prod_{i=1}^{N} q_i^{n_i}$$

$$= \sum_{\text{all } \boldsymbol{n} \text{ s.t. } C(\boldsymbol{n}, V) = m, n_N = 0} \prod_{i=1}^{N} q_i^{n_i} + q_N \sum_{\text{all } \boldsymbol{n} \text{ s.t. } C(\boldsymbol{n}, V) = m - v_N, n_N = 0} \prod_{i=1}^{N} q_i^{n_i}.$$

Alternatively, we can write

$$h(V, m) = h^{-N}(V, m) + q_N h^{-N}(V, m - v_N) \quad (7)$$

where $h^{-N}(V, m)$ is the same as $h(V, m)$ except that it is evaluated for a system with node N removed, i.e.,

$$h^{-N}(V, m) = \sum_{\text{all } \boldsymbol{n} \text{ s.t. } C^{-N}(\boldsymbol{n}, V) = m} \prod_{i=1}^{N-1} q_i^{n_i}$$

where $C^{-N}(\boldsymbol{n}, V) = \sum_{i=1}^{N-1} n_i v_i$.

For a given V, the algorithm given in Fig. 3 that is derived from (7) can be used to evaluate $H(m) = h(V, m)$.

If the vote assignment and read quorum of each read set in Ω_N is given, the algorithm described in Fig. 3 can be used to compute $\alpha(P, V)$ for each (V, r) corresponding to read sets in Ω_N. The (V, r) that yields the highest value for $\alpha(P, V)$ is the optimal vote and quorum assignment. However, if the nodes are labeled such that $p_1 \leq p_2 \leq \cdots \leq p_N$, we need only consider the nondecreasing vote assignments which correspond to the members of E_N generated by Algorithm 2. In other words, the members in $\Omega_N - E_N$ need not be considered for optimizing availability since they are clearly suboptimal. This follows from the intuitive idea that the best vote assignment is the one that assigns more votes to nodes with higher availability which is stated as the following theorem. Note, however, that for a different performance measure, all elements of Ω_N may have to be considered.

Theorem 3.1: Given an availability vector P and a vote assignment V, where, without loss of generality, $p_1 \leq p_2 \leq \cdots \leq p_N$, $v_1 \leq v_2 \leq \cdots \leq v_N$. Then,

$$\alpha(P, V) \geq \alpha(P, V')$$

for any permutation V' of the vote assignment V.

Proof: See Appendix B.

IV. Numerical Examples

We demonstrate the use of the results derived in the previous sections by analyzing systems of five and seven nodes. For these systems, we show the optimal vote and quorum assignment and the resulting system availability.

[3]This is the same technique used to define the basic relationship for the convolution algorithm used to evaluate closed queueing networks [18].

```
Initialize:  H(0) := 1; H(1), H(2), . . ., H(L) := 0;
for i := 1 to N do
    for m := L downto v_i do
        H(m) := H(m) + q_i * H(m - v_i)
    end;
end;
```

Fig. 3. Algorithm for computing $h(V, m)$.

TABLE III
Best Read/Write Coteries for Two Systems of Five Nodes

	$p_i = 0.9$, others = 0.8		$p_1 = p_2 = 0.9$, others = 0.8	
f	Vote assignment	r	Vote assignment	r
0.001	(1, 1, 1, 1, 1)	5	(1, 1, 1, 1, 1)	5
0.1	(1, 1, 1, 1, 1)	4	(1, 1, 1, 2, 2)	5
0.2	(1, 1, 1, 1, 2)	4	(1, 1, 1, 2, 2)	5
0.3	(1, 1, 1, 1, 2)	4	(2, 2, 2, 3, 3)	7
0.4	(1, 1, 1, 1, 1)	3	(2, 2, 2, 3, 3)	7
0.5	(1, 1, 1, 1, 1)	3	(1, 1, 1, 2, 2)	4
0.6	(1, 1, 1, 1, 1)	3	(2, 2, 2, 3, 3)	6
0.7	(1, 1, 1, 1, 2)	3	(2, 2, 2, 3, 3)	6
0.8	(1, 1, 1, 1, 2)	3	(1, 1, 1, 2, 2)	3
0.9	(1, 1, 1, 1, 1)	2	(1, 1, 1, 2, 2)	3
0.999	(1, 1, 1, 1, 1)	1	(1, 1, 1, 1, 1)	1

Table III shows the optimal vote and quorum assignments for two systems of five nodes for various values of the transaction mix f. Since the system considered in the first column is relatively homogeneous (four nodes have the same availability which is 0.8 and the availability of the other node is 0.9), either the uniform vote assignment is optimal (each replica gets one vote) or the node with higher availability is assigned an extra vote. This does not hold, as shown in column two of the table, when the availability of two nodes is 0.9. In this case, the optimal vote assignment is not uniform for all cases except when f is close to 0 or 1.

We show the system availability in Fig. 4 for the vote and quorum assignments of Table III when the availability of three nodes is 0.8 and it is 0.9 for the other two nodes. Clearly, no single vote and quorum assignment can provide optimal availability for all values of f. For example, (2, 2, 2, 3, 3) with $r = 6$ is optimal for $f = 0.6$ but when f changes to 0.8, the optimal assignment becomes (1, 1, 1, 2, 2) with $r = 3$. Also, note that the availability is not very sensitive to a change in f for some of the vote and quorum assignments.

The optimal system availability as a function of f for the two systems considered above and one where the availability of each node is 0.8, is shown in Fig. 5. The plot for the system of Fig. 5 is obtained from the plots of Fig. 4 by taking the highest system availability for a given f. Although the optimal availabilities of these systems are similar when f is close to 0 or 1, for other values of f, α increases by almost 2 percent when the availability of one node changes from 0.8 to 0.9. Also, the optimal availability for each of the systems is not sensitive to a change in f when f is not close to 0 or 1.

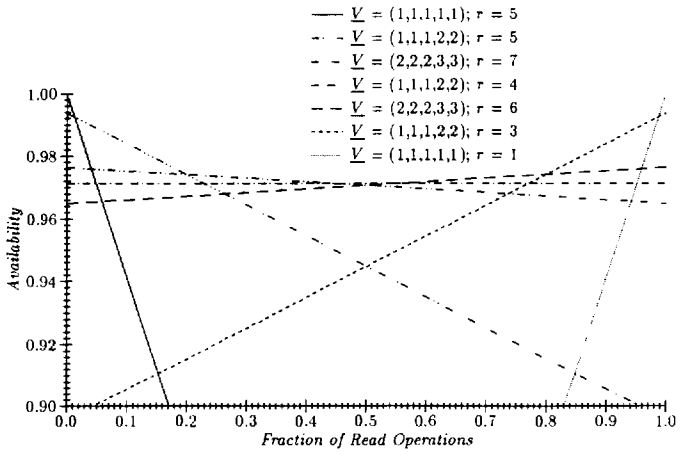


Fig. 4. Read/write availability of read coteries, $p_1 = p_2 = p_3 = 0.8$, other nodes = 0.9.

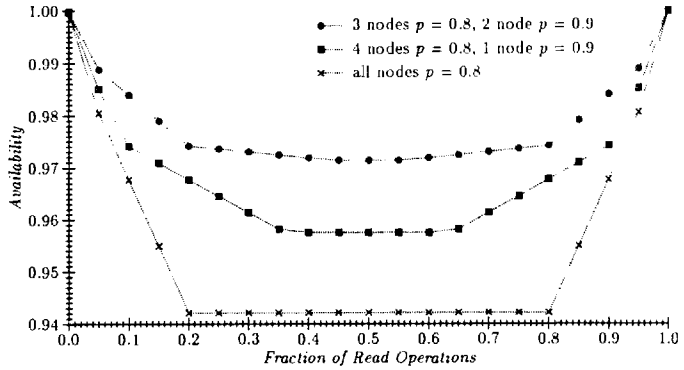


Fig. 5. Optimal read/write availability for three systems of five nodes.

To further illustrate how the optimal vote assignment depends on the node availabilities and f, in Table IV we also consider a system of seven nodes, where the availability of each node is different ($p_1 = 0.65$, $p_2 = 0.7$, $p_3 = 0.75$, $p_4 = 0.8$, $p_5 = 0.85$, $p_6 = 0.9$, and $p_7 = 0.95$). For $f = 0.9$, we see that the most available node is assigned seven votes while the least available node gets only one vote. For $f = 0.8$, the difference in the votes is even more marked.

Finally, to show that the availability obtained from optimal vote and quorum assignment can be appreciably higher than that of the commonly used quorums with a uniform vote assignment (i.e., when $v_i = 1$ for all i), we consider the data in Table V. In the system, two nodes are highly available ($p = 0.9$) while the others only have an availability of 0.6. We see that when $f = 0.8$, the optimal availability is about 9 percent higher compared to the read majority/write majority quorum. Even when the optimal quorum, r_{opt}, is considered for a uniform vote assignment in this system, the availability is still 5.5 percent lower than that achievable with optimal vote and quorum assignment. The read one/write all quorum has poorer availability for all values of f except when f is 0.999. Thus, compared to the commonly used quorums with each node having a single vote, the optimal vote and quorum assignment provides better system availability.

TABLE IV
BEST READ/WRITE SETS FOR A SYSTEM OF SEVEN NODES WITH AVAILABILITY VECTOR = (0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95)

f	Vote assignment	r
0.001	(1, 1, 1, 2, 2, 3, 3)	11
0.1	(1, 2, 2, 3, 4, 5, 7)	15
0.2	(3, 4, 5, 6, 8,10,13)	28
0.3	(2, 3, 4, 5, 6, 8,10)	21
0.4	(2, 2, 3, 4, 5, 6, 8)	16
0.5	(1, 2, 3, 3, 4, 5, 7)	13
0.6	(2, 2, 3, 4, 5, 6, 8)	15
0.7	(2, 3, 4, 5, 6, 8,10)	18
0.8	(3, 4, 5, 6, 8,10,13)	22
0.9	(1, 2, 2, 3, 4, 5, 7)	10
0.999	(1, 1, 1, 2, 2, 3, 3)	3

TABLE V
OPTIMIZING VOTE AND QUORUM VERSUS OPTIMIZING QUORUM USING UNIFORM VOTE ASSIGNMENT FOR A SYSTEM OF SEVEN NODES WITH AVAILABILITY VECTOR (0.6, 0.6, 0.6, 0.6, 0.6, 0.9, 0.9)

$\alpha^{(1)}$ = optimum availability over all vote assignments and quorums
$\alpha^{(2)}$ = availability of the read one/write all quorum
$\alpha^{(3)}$ = availability of the read majority/write majority quorum
$\alpha^{(4)}$ = optimum availability using the uniform vote assignment

	All Vote Assignments			Uniform Vote Assignment			
f	Optimal $\underline{V}_N, r$		$\alpha^{(1)}$	$\alpha^{(2)}$	$\alpha^{(3)}$	r_{opt}	$\alpha^{(4)}$
0.001	(1, 1, 1, 1, 1, 1, 1)	7	0.999	0.064	0.866	7	0.999
0.1	(0, 0, 0, 0, 0, 1, 1)	2	0.972	0.157	0.866	5	0.937
0.2	(1, 1, 1, 1, 1, 5, 5)	10	0.955	0.250	0.866	5	0.901
0.3	(1, 1, 1, 1, 1, 4, 4)	8	0.943	0.344	0.866	4	0.866
0.4	(1, 1, 1, 1, 1, 3, 3)	6	0.933	0.438	0.866	4	0.866
0.5	(1, 1, 1, 1, 1, 3, 3)	6	0.933	0.531	0.866	4	0.866
0.6	(1, 1, 1, 1, 1, 3, 3)	6	0.933	0.625	0.866	4	0.866
0.7	(1, 1, 1, 1, 1, 4, 4)	6	0.943	0.719	0.866	4	0.866
0.8	(1, 1, 1, 1, 1, 5, 5)	6	0.955	0.813	0.866	3	0.901
0.9	(0, 0, 0, 0, 0, 1, 1)	1	0.972	0.906	0.866	3	0.937
0.999	(1, 1, 1, 1, 1, 1, 1)	1	0.999	0.999	0.866	1	0.999

V. CONCLUDING REMARKS

We have presented a method for obtaining an enumeration of vote assignable read coteries and their corresponding vote assignments and quorums. We have also developed an efficient method for finding the availability for any vote assignment when the availability of the nodes and the read/write transaction mix are given. The use of this method was demonstrated by finding the vote assignment and quorum that yields the highest system availability in systems with different node availabilities. It should be emphasized that the enumeration of the vote assignable read coteries can also be used in the optimization of other performance measures. In future research, we plan to study the effect of the communication network and the performance of dynamic voting schemes. We will also consider direct methods for finding optimal settings in voting that maximizes system availability for reading and writing that can be used for optimizing larger systems.

APPENDIX A

This Appendix describes the proofs of the lemmas used in Section II.

Lemma A.1—Removing a Replica from a Read Group: Let $V_N = (v_1, v_2, \cdots, v_N)$ be a vote assignment to N replicas. Let $G = \{g_1, g_2, \cdots, g_k\}$ be a read group in $Q_N(r, V_N)$. Then, $\forall g_i \in G: G - \{g_i\} \in Q_N(r - v_{g_i}, V_N)$.

Proof: Since $v(G) \geq r$, $v(G - \{g_i\}) \geq r - v_{g_i}$ for any $g_i \in G$. The only reason that $G - \{g_i\} \notin Q_N(r - v_{g_i}, V_N)$ is when it is not tight, i.e., there is a replica $g_k \in G$ such that $v(G - \{g_i\}) \geq r - v_{g_i} + v_{g_k}$. However, this implies that $v(G) \geq r + v_{g_k}$ which is a contradiction since G is tight.

Lemma A.2—Adding a Replica to a Read Group: Let $V_N = (v_1, v_2, \cdots, v_N)$, $V_{N+1} = (v_1, v_2, \cdots, v_{N+1})$ and let $G \in Q_N(r, V_N)$. Then:

If $v(G) \geq r + v_{N+1}$, then $G \in Q_{N+1}(r + v_{N+1}, V_{N+1})$, otherwise, $G \cup \{N+1\} \in Q_{N+1}(r + v_{N+1}, V_{N+1})$.

Proof: If $v(G) \geq r + v_{N+1}$, then G is also a tight group of $r + v_{N+1}$ votes, or $G \in Q_{N+1}(r + v_{N+1}, V_{N+1})$.

If $v(G) < r + v_{N+1}$, then G does not have a sufficient number of votes to be a group of $Q_{N+1}(r + v_{N+1}, V_{N+1})$. The group $G \cup \{N+1\}$ will have at least $r + v_{N+1}$ votes and to show that it is tight, let g_1 be the replica in G with the least number of votes. If $v_{N+1} \leq v_{g_1}$, then $N+1$ is the replica with the least number of votes in $G \cup \{N+1\}$ and removal of $N+1$ from $G \cup \{N+1\}$ leaves a group of less than $r + v_{N+1}$ votes and thus G is tight. If $v_{N+1} > v_{g_1}$, then $G \cup \{N+1\}$ satisfies

$$\begin{aligned} r + v_{N+1} &\leq v(G \cup \{N+1\}) \\ &\leq (r + v_{N+1} - 1) + v_{g_1} \end{aligned}$$

because G satisfies $r \leq v(G) \leq (r-1) + v_{g_1}$. Therefore, $G \cup \{N+1\}$ is tight and $G \cup \{N+1\} \in Q_{N+1}(r + v_{N+1}, V_{N+1})$. □

Lemma 2.1—Expanding the Vote Assignment: Let vote assignments V_N and V_{N+1} be as in Lemma A.2. Then,

$$\begin{aligned} &Q_{N+1}(r, V_{N+1}) \\ &\quad = Q_N(r, V_N) \cup \\ &\qquad \{G \cup \{N+1\} \mid G \in Q_N(r - v_{N+1}, V_N) \\ &\qquad \wedge\, G \notin Q_N(r, V_N)\}. \end{aligned}$$

Proof: (By showing LHS ⊆ RHS and RHS ⊆ LHS.)

We first show that LHS ⊆ RHS. Let $H \in Q_{N+1}(r, V_{N+1})$. If $N+1 \notin H$, then $H \in Q_N(r, V_N)$. When $N+1 \in H$, then $H \notin Q_N(r, V_N)$. Also, $H - \{N+1\} \notin Q_N(r, V_N)$ because H is a tight group of r votes, and hence $v(H - \{N+1\}) < r$. Define the group $G_1 = H - \{N+1\}$. We have, by Lemma A.1, that $G_1 \in Q_{N+1}(r - v_{N+1}, V_{N+1})$, and since $N+1 \notin G_1$, $G_1 \in Q_N(r - v_{N+1}, V_N)$. Hence, $H \in \{G_1 \cup \{N+1\} \mid G_1 \in Q_N(r - v_{N+1}, V_N) \wedge G_1 \notin Q_N(r, V_N)\}$.

We can show that RHS ⊆ LHS by showing that each of the sets in the RHS are subsets of the LHS. When $H \in Q_N(r, V_N)$, it must follow that $H \in Q_{N+1}(r, V_{N+1})$. Now let $H = G_1 \cup \{N+1\}$, where $G_1 \in Q_N(r - v_{N+1}, V_N)$ and $G_1 \notin Q_N(r, V_N)$. Then, by Lemma A.2, either $G_1 \in Q_{N+1}(r, V_{N+1})$ or $G_1 \cup \{N+1\} \in Q_{N+1}(r, V_{N+1})$. The former case is not possible because $G_1 \notin Q_N(r, V_N)$ and $N+1 \notin G_1$, then $G_1 \notin Q_{N+1}(r, V_{N+1})$. Thus, $H = G_1 \cup \{N+1\} \in Q_{N+1}(r, V_{N+1})$. □

When two read coteries $Q_N(r, V_N)$ and $Q_N(s, W_N)$ are isomorphic, a permutation π maps $Q_N(r, V_N)$ to $Q_N(s, W_N)$. The replica $\pi(i)$ in $Q_N(s, W_N)$ thus plays the role of the replica i in $Q_N(r, V_N)$. The role of a replica is determined by its votes and it follows that π is dependent on the assignments V_N and W_N. The following two lemmas show that when V_N and W_N are nondecreasing, the read coteries $Q_N(r, V_N)$ and $Q_N(s, W_N)$ are isomorphic if and only if they are equal. In Lemma A.3, we consider the case when two read coteries are mapped to each other by a transposition (a transposition is a permutation that exchanges two elements) and in Lemma 2.2 we consider the general case.

Lemma A.3—Equality of Isomorphic Read Coteries under a Transposition: Let V_N and W_N be two vote assignments such that for some $a < b$, $v_a \leq v_b$, and $w_a \leq w_b$ and let τ be the transposition $(a\ b)$. Then, for some r and s, $Q_N(r, V_N)$ is isomorphic to $Q_N(s, W_N)$ under τ if and only if $Q_N(r, V_N) = Q_N(s, W_N)$.

Proof: By contradiction.

Assume that $Q_N(r, V_N)$ and $Q_N(s, W_N)$ are isomorphic under τ but not equal. It cannot be the case that $Q_N(r, V_N) \subset Q_N(s, W_N)$ or $Q_N(s, W_N) \subset Q_N(r, V_N)$ because isomorphic sets have equal number of groups. Therefore, there is a group $G \in Q_N(r, V_N)$ such that $G \notin Q_N(s, W_N)$. We split the proof into cases depending on whether a and/or b is an element of G. Write G as $G_1 \cup H$ where G_1 does not contain a and b, and $H \subseteq \{a, b\}$.

H cannot be ϕ or $\{a, b\}$, since if $G = G_1$ or $G = G_1 \cup \{a, b\}$, then $G \in Q_N(s, W_N)$ contradicting the assumption that $G \notin Q_N(s, W_N)$.

Case 1: $G = G_1 \cup \{a\}$.

If $G_1 \cup \{b\} \in Q_N(r, V_N)$ also, then $G_1 \cup \{a\} \in Q_N(s, W_N)$, contradicting that $G \notin Q_N(s, W_N)$. So it must be that $G_1 \cup \{b\} \notin Q_N(r, V_N)$ and the following chains of implications can be made from the fact that $G_1 \cup \{a\} \in Q_N(r, V_N)$ and $G_1 \cup \{b\} \notin Q_N(r, V_N)$:

$$\begin{aligned} &G_1 \cup \{a\} \in Q_N(r, V_N) \wedge \\ &\qquad G_1 \cup \{b\} \notin Q_N(r, V_N) \\ &\Rightarrow G_1 \cup \{b\} \in Q_N(s, W_N) \wedge \\ &\qquad G_1 \cup \{a\} \notin Q_N(s, W_N) \\ &\Rightarrow \forall g \in G_1: v(G_1) + w_b - w_g < s \wedge v(G_1) < s \\ &\qquad [G_1 \cup \{b\} \text{ is tight}] \end{aligned}$$

$$\Rightarrow \forall g \in G_1 : v(G_1) + w_a - w_g < s \wedge v(G_1) < s$$
$$[w_a \leq w_b]$$
$$\Rightarrow \text{if } v(G_1 \cup \{a\}) \geq s, \text{ then}$$
$$G_1 \cup \{a\} \in Q_N(s, W_N)$$
$$\Rightarrow v(G_1 \cup \{a\}) < s$$
$$[G_1 \cup \{a\} \notin Q_N(s, W_N)].$$

And,

$$G_1 \cup \{a\} \in Q_N(r, V_N) \wedge$$
$$G_1 \cup \{b\} \notin Q_N(r, V_N)$$
$$\Rightarrow v(G_1 \cup \{b\}) \geq v(G_1 \cup \{a\}) \geq r$$
$$[v_b \geq v_a]$$
$$\Rightarrow \text{sub}(G_1 \cup \{b\}) \in Q_N(r, V_N)$$
$$[G_1 \cup \{b\} \notin Q_N(r, V_N)]$$
$$\Rightarrow \text{sub}(G_1) \cup \{b\} \in Q_N(r, V_N)$$
$$[G_1 \cup \{a\} \in Q_N(r, V_N)]$$
$$\Rightarrow \text{sub}(G_1) \cup \{a\} \in Q_N(s, W_N)$$
$$\Rightarrow v(\text{sub}(G_1) \cup \{a\}) \geq s$$
$$\Rightarrow v(G_1 \cup \{a\}) > s$$

contradicting the previous conclusion.

The case for $G = G_1 \cup \{b\}$ is similar to the previous one. If $G_1 \cup \{a\} \in Q_N(r, V_N)$ also, then $G_1 \cup \{b\} \in Q_N(s, W_N)$ which contradicts that $G \notin Q_N(s, W_N)$. So it must be that $G_1 \cup \{a\} \notin Q_N(r, V_N)$ and as in the previous case, we can derive contradicting conclusions that $v(G_1 \cup \{a\}) < r$ and $v(G_1 \cup \{a\}) > r$.

Lemma 2.2—Equality of Isomorphic Read Coteries of Nondecreasing Vote Assignments: Let $V_N = (v_1, v_2, \cdots, v_N)$ and $W_N = (w_1, w_2, \cdots, w_N)$ be two nondecreasing vote assignments. $Q_N(r, V_N)$ and $Q_N(s, W_N)$ are isomorphic if and only if $Q_N(r, V_N) = Q_N(s, W_N)$.

Proof: We will use induction to prove that if $Q_N(r, V_N)$ and $Q_N(s, W_N)$ are isomorphic then they are equal. The converse is obvious.

Let π be a permutation that maps $Q_N(r, V_N)$ to $Q_N(s, W_N)$, i.e., $Q_N(s, W_N) = \pi(Q_N(r, V_N))$ where $\pi(Q_N(r, V_N))$ denotes the read coterie obtained by replacing i by $\pi(i)$ in the read coterie $Q_N(r, V_N)$. From the theory of permutations (see for example [19]), we know that a permutation can be factorized into disjoint cycles and this factorization is unique except for the order in which the cycles are written. Also, each l-cycle $(i_1\ i_2\ \cdots\ i_l)$ can be written as a product of $l - 1$ transpositions as follows: $(i_l\ i_1)\ (i_{l-1} i_1) \cdots (i_2\ i_1)$. Therefore, if a permutation π can be factorized into c cycles and cycle i is of length l_i, then π can be factorized into $\Sigma_{i=1}^{c}(l_i - 1)$ transpositions. The proof of the lemma is by induction on the number of transpositions that constitute the factorization of π. For clarity, we use τ to denote a transposition.

Basis: $\pi = \tau_1 = (a\ b)$.

Without loss of generality, assume $a < b$. Then,

$$Q_N(s, W_N) = \tau_1(Q_N(r, V_N)).$$

We thus have by Lemma A.3 that $Q_N(s, W_N) = Q_N(r, V_N)$ and the basis is proved.

Induction Hypothesis: Given that V_N and W_N are nondecreasing, if $Q_N(r, V_N)$ and $Q_N(s, W_N)$ are isomorphic under a permutation π_k that has a factorization of k or less transpositions, then $Q_N(r, V_N)$ and $Q_N(s, W_N)$ are equal.

Induction Step: We need to show that when V_N and W_N are nondecreasing and $Q_N(r, V_N)$ and $Q_N(s, W_N)$ are isomorphic under a permutation π_{k+1} that can be factorized into $k + 1$ transpositions, then $Q_N(r, V_N) = Q_N(s, W_N)$.

Let a be the smallest index such that $\pi_{k+1}(i) = i$; $i < a$ and $\pi_{k+1}(a) \neq a$. We can write π_{k+1} as $\tau_{k+1}\tau_k \cdots \tau_1$ such that $\tau_{k+1} = (a\ b)$ for some $b > a$ and τ_j does not move a for $j = 1, 2, \cdots, k$. This can be done by factorizing τ_{k+1} into disjoint cycles and reordering the cycles such that the cycle containing a is the left-most cycle. After rotating the left-most cycle a number of times such that the last element in the cycle is a, the formula $(i_1\ i_2 \cdots i_l) = (i_l\ i_1)\ (i_{l-1} i_1) \cdots (i_2\ i_1)$ can be used to factorize each cycle to obtain the desired τ_j, $j = 1, 2, \cdots, N + 1$.

Denote $\tau_k\tau_{k-1} \cdots \tau_1$ by π_k. The permutation π_{k+1} can thus be written as a product of the transposition $\tau_{k+1} = (a\ b)$ and the permutation τ_k that has a factorization of k transpositions. Note that the transpositions τ_j, $j = 1, 2, \cdots, k$ do not move the replicas $1, 2, \cdots, a$ and hence $\pi_k(i) = i$ for $i = 1, 2, \cdots, a$. We can write

$$\begin{aligned} Q_N(s, W_N) &= \pi_{k+1}(Q_N(r, V_N)) \\ &= \tau_{k+1} \cdot \pi_k(Q_N(r, V_N)) \\ &= \tau_{k+1}(Q_N(r, U_N)) \end{aligned}$$

where U_N is a vote assignment such that $u_i = v_{\pi_k(i)}$. Since π_k does not move $1, 2, \cdots, a$, $u_a = v_a$ and since for all $i > a$, $v_i \geq v_a$, we have $u_b \geq u_a$. By Lemma A.3, we conclude that $Q_N(s, W_N) = Q_N(r, U_N) = \pi_k(Q_N(r, V_N))$, in other words, $Q_N(r, V_N)$ and $Q_N(s, W_N)$ are isomorphic under a permutation that can be factorized into k transpositions. By the induction hypothesis, we can conclude that $Q_N(s, W_N) = Q_N(r, V_N)$. □

APPENDIX B

Theorem 3.1: Given an availability vector P and a vote assignment V where, without loss of generality, $p_1 \leq p_2$

$\leq \cdots \leq p_N$, $v_1 \leq v_2 \leq \cdots \leq v_N$. Then,

$$\alpha(\boldsymbol{P}, \boldsymbol{V}) \geq \alpha(\boldsymbol{P}, \boldsymbol{V}')$$

for any permutation $\boldsymbol{V}'$ of the vote assignment $\boldsymbol{V}$.

Proof: We prove the theorem by showing that a vote assignment can be improved by exchanging the votes between two nodes if one of them has higher availability but is assigned less votes, that is:

Given availability vector $\boldsymbol{P}$ where, without loss of generality, $p_1 \leq p_2 \leq \cdots \leq p_N$. For the two vote assignments

$$\boldsymbol{V} = (v_1, v_2, \cdots, v_N)$$

and

$$\boldsymbol{V}' = (v_1', v_2', \cdots, v_N')$$

such that for some value l and k:

$$1 \leq l < k \leq N$$

$$v_i' = v_i \quad \text{for } i \neq k, l;$$

$$v_k' = v_l;\ v_l' = v_k \text{ and } v_l \leq v_k$$

we have

$$\alpha_s(\boldsymbol{P}, \boldsymbol{V}) \geq \alpha_s(\boldsymbol{P}, \boldsymbol{V}')$$

where s is an arbitrary threshold and $\alpha_s(\boldsymbol{P}, \boldsymbol{V})$ is given by (6).

From (6) we have

$$\alpha_s(\boldsymbol{P}, \boldsymbol{V}) = \frac{1}{K} \sum_{m=s}^{L} \sum_{\text{all } \boldsymbol{n} \text{ s.t. } C(\boldsymbol{n}, \boldsymbol{V}) = m} \prod_{i=1}^{N} q_i^{n_i}. \quad (8)$$

We define

$$h^{-lk}(\boldsymbol{V}, m) = \sum_{\text{all } \boldsymbol{n} \text{ s.t. } C^{-lk}(\boldsymbol{n}, \boldsymbol{V}) = m} \prod_{i=1, i \neq l, k}^{N} q_i^{n_i} \quad (9)$$

where $C^{-lk}(\boldsymbol{n}, \boldsymbol{V}) = \sum_{i=1, i \neq l, k}^{N} n_i v_i$. We thus have that

$$K\alpha_s(\boldsymbol{P}, \boldsymbol{V}) = \sum_{m=s}^{L} \left[h^{-lk}(\boldsymbol{V}, m) + q_l h^{-lk}(\boldsymbol{V}, m - v_l) + q_k h^{-lk}(\boldsymbol{V}, m - v_k) + q_l q_k h^{-lk}(\boldsymbol{V}, m - v_l - v_k) \right]. \quad (10)$$

Similarly, we get

$$K\alpha_s(\boldsymbol{P}, \boldsymbol{V}') = \sum_{m=s}^{L} \left[h^{-lk}(\boldsymbol{V}, m) + q_k h^{-lk}(\boldsymbol{V}, m - v_l) + q_l h^{-lk}(\boldsymbol{V}, m - v_k) + q_l q_k h^{-lk}(\boldsymbol{V}, m - v_l - v_k) \right]. \quad (11)$$

Therefore, we have

$$K(\alpha_s(\boldsymbol{P}, \boldsymbol{V}) - \alpha_s(\boldsymbol{P}, \boldsymbol{V}')) = (q_k - q_l) \sum_{m=s}^{L} \left[h^{-lk}(\boldsymbol{V}, m - v_k) - h^{-lk}(\boldsymbol{V}, m - v_l) \right]. \quad (12)$$

If $p_l \leq p_k$ and $q_i = p_i/(1 - p_i)$, we also have $q_l \leq q_k$. To prove the lemma, it is thus sufficient to show that

$$\sum_{m=s}^{L} \left[h^{-lk}(\boldsymbol{V}, m - v_k) - h^{-lk}(\boldsymbol{V}, m - v_l) \right] \geq 0. \quad (13)$$

The left-hand side of (13) can be written as

$$\sum_{m=s-v_k}^{L-v_k} h^{-lk}(\boldsymbol{V}, m) - \sum_{m=s-v_l}^{L-v_l} h^{-lk}(\boldsymbol{V}, m). \quad (14)$$

Since $v_l \leq v_k$, then also $s - v_l \geq s - v_k$ and $L - v_l \geq L - v_k$. Cancel out the identical terms in (14) and we thus get (14) equal to

$$\sum_{m=s-v_k}^{s-v_l-1} h^{-lk}(\boldsymbol{V}, m) - \sum_{m=L-v_k+1}^{L-v_l} h^{-lk}(\boldsymbol{V}, m). \quad (15)$$

Note that $h^{-lk}(\boldsymbol{V}, m) \geq 0$ for all m and $h^{-lk}(\boldsymbol{V}, m) = 0$ for $m > L - v_l - v_k$ (because when replicas j and k are removed, the system is left with $L - v_l - v_k$ votes). The right sum of (15) is equal to 0, and hence (15) is greater than or equal to 0.

References

[1] S. B. Davidson, H. Garcia-Molina, and D. Skeen, "Consistency in partitioned network," *ACM Comput. Survey*, vol. 17, no. 3, pp. 341–370, 1985.

[2] H. Gifford, "Weighted voting for replicated data," in *Proc. 7th Symp. Operat. Syst.*, 1979, pp. 150–162.

[3] D. Eager and K. Sevcik, "Achieving robustness in distributed database systems," *ACM Trans. Database Syst.*, vol. 8, no. 3, pp. 354–381, 1983.

[4] M. Herlihy, "Dynamic quorum adjustments for partitioned data," Carnegie-Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-CS-86-147, 1987.

[5] A. E. Abbadi and S. Toueg, "Maintaining availability in partitioned replicated databases," in *Proc. Symp. Principles Database Syst. (PODS)*, 1986, pp. 240–351.

[6] S. Jajodia and D. Mutchler, "Dynamic voting," in *Proc. SIGMOD-87*, 1987, pp. 227–238.

[7] D. Barbara, H. Garcia-Molina, and A. Spauster, "Protocols for dynamic vote reassignment," in *Proc. Principles of Distributed Comput.*, 1986, pp. 195–205.

[8] J. Paris, "Voting with witnesses: A consistency scheme for replicated files," in *Proc. 6th Int. Conf. Distributed Comput. Syst.*, 1986, pp. 606–612.

[9] D. Davcev and W. Burkhard, "Consistency and recovery control for replicated data," in *Proc. 10th Symp. Operat. Syst. Principles*, 1985, pp. 87–96.

[10] D. Agrawal and A. E. Abbadi, "Reducing storage for quorum consensus algorithms," in *Proc. Very Large Databases Conf.*, 1988, pp. 419–430.

[11] H. Garcia-Molina and D. Barbara, "How to assign votes in a distributed system," *J. ACM*, vol. 32, no. 4, pp. 841–860, 1985.

[12] L. Lamport, "The implementation of reliable distributed multiprocess systems," *Comput. Networks*, vol. 2, pp. 95–114, 1978.

Paper # 5

On the Performance of Protocols for Collecting Responses over a Multiple-Access Channel *

Mostafa H. Ammar and George N. Rouskas
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332

Abstract

We consider a generalization of the multiple access problem where it is necessary to identify a subset of the ready users, not all. The problem is motivated by several "response collection" applications that arise in distributed computing and database systems. In these applications, a collector is interested in gathering a set of responses from a number of potential respondents. The collector and respondents communicate over a shared channel. We define three collection objectives and investigate a suite of protocols that can be used to achieve these objectives. The protocols are based on the use of polling, TDMA, and group testing. Using a binomial respondent model we analyze and, where applicable, optimize the performance of the protocols. Our concern is with cost measures that reflect the computational load placed on the system, as well as the delay incurred for achieving a particular objective.

1 Introduction

We investigate the problem of how to best collect a specified number of responses from a set of nodes over a multiple access channel. Several situations in distributed systems where such a problem arises are described later. We consider a system where nodes share a common communication channel. One node in the system is interested in collecting responses from the other nodes. Not all nodes can or will respond when requested and the node soliciting responses is interested in achieving a collection objective.

The problem we consider is actually a generalization of the multiple access communication problem where we are concerned with identifying a *subset* of ready users, not all. A response collection process will be aimed at achieving one of a set of *collection objectives* to be described later. We describe and analyze a suite of protocols that can be used for response collection. Our concern is with the cost of the collection process in terms of the amount of computation resources it consumes, as well as the amount of time expended to achieve a certain collection objective. The protocols we use are based on the use of polling, time division multiple access (TDMA) and group testing.

Whereas polling and TDMA are well known multiple access techniques, group testing warrants a short introduction. It is a technique that can be used to efficiently identify "defective" items in a set. It has been studied extensively in different contexts (see for example [1, 2, 3, 4]). The basic idea of the technique is the testing of items being inspected in groups. The composition of the group to be tested at any one point in time being dictated by the history of previous test outcomes. Each test is counted as a single step and the objective is to determine group composition rules to minimize the number of steps. In its original form, the problem assumes the outcome of each test would indicate one of two situations: "all items are not defective" or "there is at least one defective item." We are concerned here with the potential use of group testing as a technique for collision resolution over a multiple access channel. Such use has been described in [5, 6, 7, 8]. The additional feature when using group testing over a multiple-access channel is the ability to differentiate among three possible outcomes when a group is enabled: no transmission, one transmission, and more than one transmission (a collision).

This paper is organized as follows. In section 2 we discuss some applications that motivate our work. Section 3 contains a model of our system. Section 4 presents a description and analysis of the Polling and TDMA protocols. In section 5 we describe and analyze an approach based on the staging of the response collection process where in each stage a TDMA protocol is employed. Sections 6 and 7 investigate the group testing and staged group testing protocols. Some numerical examples are presented in section 8 and section 9 contains some concluding remarks.

2 Some Applications

The following are some applications that make use of response collection.

A database system with multiple query optimization: Here we have a shared channel LAN with the primary purpose of giving a set of attached users access to a database (also connected to the network). The users are moderately active and the database employs sophisticated query processing techniques. These include schemes to speedup query processing through the use of multiple query optimization (see e.g., [9]). Rather than processing each query individually, the database tries to process a number of queries (up to a maximum) at a time. In order to manage memory and processing resources efficiently, the database processor prefers to actively collect responses, rather than receiving responses asynchronously.

*This work is supported in part by NSF grant NCR-8604850

[13] D. Barbara and H. Garcia-Molina, "The reliability of voting mechanisms," *IEEE Trans. Comput.*, vol. C-36, no. 10, pp. 1197-1208, 1987.
[14] Z. Tong and R. Kain, "Vote assignments in weighted voting mechanisms," in *Proc. 7th Symp. Reliable Distributed Syst.*, 1988, pp. 138-143.
[15] M. Ahamad and M. Ammar, "Performance of quorum-consensus algorithms for replicated data," *IEEE Trans. Software Eng.*, vol. 15, no. 4, 1989.
[16] W. Smith and P. Decitre, "An evaluation method for analysis of the weighted voting algorithm for maintaining replicated data," in *Proc. 4th Int. Conf. Distributed Comput. Syst.*, 1984, pp. 494-502.
[17] A. F. Veinott and G. B. Dantzig, "Integral extreme points," *SIAM Rev.*, vol. 10, no. 3, pp. 371-372, 1968.
[18] J. Buzen, "Computational algorithms for closed queueing networks with exponential servers," *Commun. ACM*, vol. 16, no. 9, pp. 527-531, 1973.
[19] J. J. Rotman, *An Introduction to the Theory of Groups*. Boston: Allyn and Bacon, 1984.

Shun Yan Cheung received the Kandidaats and Ingenieur degrees from the Department of Mathematics and Computer Science, Delft Technological University, Delft, The Netherlands, in 1981 and 1984, respectively, and the M.S. degree from the School of Information and Computer Science, Georgia Institute of Technology, Atlanta, in 1987.

He is currently a Ph.D. candidate at Georgia Tech. His current research interests include computer networks, fault tolerant systems, and performance evaluation.

Mustaque Ahamad received the B.E. (Hons.) degree in electrical engineering from the Birla Institute of Technology and Science, Pilani, India, and the M.S. and Ph.D. degrees in computer science from the State University of New York, Stony Brook, in 1983 and 1985, respectively.

Since September 1985 he has been an Assistant Professor in the School of Information and Computer Science, Georgia Institute of Technology, Atlanta. His research interests include distributed operating systems, distributed algorithms, fault-tolerant systems, and performance evaluation.

Dr. Ahamad is a member of the IEEE Computer Society.

Mostafa H. Ammar (S'83-M'85) received the S.B. and S.M. degrees from the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, in 1978 and 1980, respectively, and the Ph.D. degree from the Department of Electrical Engineering, University of Waterloo, Waterloo, Ont., Canada, in 1985.

From 1980 to 1982, he worked at Bell-Northern Research, Ottawa, Ont., Canada, first as a Member of Scientific Staff, and then as Manager of Data Network Planning. Currently, he is an Assistant Professor in the School of Information and Computer Science, Georgia Institute of Technology, Atlanta. His current research interests include information delivery systems, distributed computing and database systems, and communication networks used in manufacturing environments.

Dr. Ammar is a member of the Association for Computing Machinery, Eta Kappa Nu, and the Association of Professional Engineers of the Province of Ontario.

A broadcast delivery information system: An information system using broadcast as a delivery mechanism has the potential for *shared response*, i.e., responding to several users with one transmission (see e.g., [10, 11, 12, 13]). In order to maximize the benefit of the response sharing feature, the information source needs to be aware of the information needs of a representative set of users at any one time. Thus, before transmitting responses, the information source needs to collect a set of requests from the users.

Quorum collection for synchronization in a distributed system: Quorum consensus is a general class of synchronization protocols for distributed systems [14, 15, 16, 17]. An operation may proceed to completion only if it is granted permission from a number of nodes. If mutually exclusive execution of operations is desired, e.g., as would be required when writing replicated data, then the node executing the operation needs to collect permission from a majority of nodes. Other applications, such as the reading of replicated data, may require permission from a certain number of nodes, not necessarily a majority [18]. When quorum consensus protocols are used, if a quorum cannot be collected, the operation requesting the quorum aborts. Nodes, in some situations, may not be able to grant permission when requested if they have already granted permission to another node. They may also not be able to grant permission because they have failed or are too busy.

Finding multiple instances of a named resource: An application running in a distributed system often requires access to multiple instances of a resource. The application typically knows the name or property of such a resource, and may not be aware of where the resource is physically located in the network. The searching application needs to determine a set of addresses where the resource resides [19]. Some examples of this are: 1) a node is searching for four or more processors that are lightly loaded in order to run a parallel program, 2) a node is searching for three copies of a replicated data item in order to update it, and 3) a node is searching for up to four disks with a given amount of available space to store a certain file.

3 System Model

This section describes a model of a system that captures the salient features of the request collection applications discussed in section 2.

3.1 The Collector and Respondents

The system under consideration has a node (connected to a shared channel) which is attempting to collect responses. We call this node *the collector*. The collector actively solicits responses by transmitting messages on the channel.

All the nodes that can potentially respond to a collector's request are called *respondents*. We assume there are N such respondents. This collector-respondent classification may be permanent (as in the multiple query optimizer and the shared response system discussed above) or it may be temporary (as in the other two applications.) In the latter case, the collector will abandon its role once its response collection objective has been achieved. At that time another node may assume the collector's role. As several nodes may desire to become collectors at the same time, a fair "election" protocol needs to be available for use by the nodes. In this paper we only concern ourselves with the system behaviour from the time a new collector is identified until the collector's objective is achieved.

The collector is the (perhaps temporary) master in the system and actively solicits responses from the respondents. We distinguish between the *soliciting* and *enabling* of a respondent. A respondent is solicited once it receives a message from the collector making it aware that a collection process is underway and indicating the collection objective. A respondent is enabled if the protocol rules allow it to transmit a response on the channel if indeed it can respond. A respondent can be enabled only after or at the same time as it is solicited.

The collector is assumed to operate with some statistical knowledge of the state of the respondents. In our analysis we will assume the *binomial respondent* model. At the instant the collection process begins, each respondent will transmit a response when enabled with a probability q and with probability $p = 1 - q$ a respondent will not transmit a response when enabled.

3.2 Collection Objectives

With respect to a collector's request, a respondent is classified as *active* if it will respond when given a chance (i.e., solicited and enabled). A respondent is said to be *inactive* otherwise. The goal of the collector is to identify and collect "enough" responses from active respondents to satisfy its application. Note that in some instances, the collector's goal may be achieved if it determines (from the *lack* of responses) that the desired number of responses cannot be collected. We consider three distinct collection objectives.

1. *L or Nothing:* Terminate successfully after collecting exactly L responses or abort when a determination is made that the number of active respondents is less than L.
2. *L or Maximum:* Terminate successfully after collecting L responses, or after all respondents have been given a chance to respond, whichever occurs first.
3. *L or More:* Terminate successfully if L or more responses have been collected *and* all respondents have been given a chance to respond. Abort when a determination is made that the number of active respondents is less than L.

For example, assume the number of respondents N is 20 and $L = 6$. A collector with the "6 or Nothing" objective will terminate if 6 responses have been received or it will abort the search if out of the respondents enabled a total of 15 did not transmit responses. With the "6 or More" objective, the collector will abort in the same situation above, it will, however, continue to gather responses after 6 responses have been received. In the case of a "6 or Maximum" objective the collector will terminate (before all respondents have been enabled) only if 6 responses have been collected.

We note the following equivalencies between the various objectives: (recall that N is the total number of respondents)

$$N \text{ or } Maximum \equiv 0 \text{ or } More \equiv Find\ All\ Active \quad (1)$$

We will use the superscript (y) to denote a collection objective. In this paper we present results that pertain to the L or Maximum collection objective and we use the superscript (ℓ) to indicate the ℓ or maximum objective. Results for the other objectives have been obtained and can be found in [20].

3.3 Network Environment

All communication takes place over an error-free shared channel with capabilities for single destination, multicast and broadcast addressing. Simultaneous transmissions over the shared channel result in a collision. All response packets are assumed to be of the same size and the network can operate in a slotted mode where each slot is long enough for the transmission of a response packet. Respondents are constrained to begin transmission at a slot boundary and thus all collisions are the result of the complete overlap of response packets. The channel is assumed to provide the so-called (0, 1, e) feedback where the nodes on the channel are informed whether the previous slot contained no transmissions (0), one transmission (1), or a collision (e).

3.4 Collection Costs

For a particular protocol z and a given collection objective y, we identify three types of costs incurred in the collection process:

1. Delay Cost $D_z^{(y)}$: The average number of response slots needed until the collection objective is achieved or until a determination is made that the collection objective is not attainable.

2. Respondent Solicitation Cost $S_z^{(y)}$: The average number of respondents that are solicited in the collection process. As each solicitation message received requires interpretation and perhaps the generation of a response this measures the computation cost incurred by the respondents.

3. Collector Solicitation Cost $C_z^{(y)}$: The average number of solicitation messages sent by the collector during the collection process. This is a measure of the computation cost incurred by the collector, as well as the delay incurred each time the collector needs to send a solicitation message.

The *total cost* incurred by collection protocol z with objective y is given by:

$$A_z^{(y)} = \alpha D_z^{(y)} + \beta S_z^{(y)} + \gamma C_z^{(y)} \tag{2}$$

where α, β, and γ are weights assigned to the various costs. We also define the *total solicitation cost*

$$B_z^{(y)} = \beta S_z^{(y)} + \gamma C_z^{(y)} \tag{3}$$

We will drop the subscript describing the protocol when it is clear from the context to which protocol the quantity refers.

4 Polling and TDMA

4.1 Description

The collector may employ several techniques to achieve its objective. One possible approach is to poll all respondents individually. Each polling message sent by the collector solicits and enables one respondent. The major disadvantage of the polling approach is that it may require a significant amount of time to complete as each poll requires two messages to be sent if the outcome is positive (i.e., a response is generated by the respondent) or a message followed by a timeout period if the outcome is negative.

Another approach which would require less time is for the collector to declare its objective to the entire network via a broadcast message and have the active respondents send their responses. If the responses are transmitted using a random access scheme, a considerable amount of time and bandwidth may be wasted until the required number of respondents successfully transmit their responses. Alternatively, the protocol may operate by having the respondents ordered in some (perhaps random) way and allocate a time slot to each respondent. Active respondents transmit their responses in the allocated slot. Slots allocated to inactive respondents remain idle. As all respondents can hear channel activity, they all know when the collection objective (declared by the collector in its broadcast message) has been achieved and this collection phase is terminated. We call this technique, *the TDMA collection protocol.* Note that in this scheme the respondents are all solicited by the initial broadcast message. A respondent is enabled at the beginning of its allocated slot.

The TDMA protocol will achieve the collector's objective in less time than a polling procedure. The TDMA protocol, on the other hand, will involve all the respondents (whether or not they are active) as they will receive the initial broadcast message which will have to be interpreted by all the receiving hosts.

4.2 Analysis

For the polling protocol we let $R^{(y)}$ be the average number of respondents that need to be polled before collection objective (y) is achieved. We have that $D_{polling}^{(y)} = C_{polling}^{(y)} = S_{polling}^{(y)} = R^{(y)}$.

For the L *or Maximum* collection objective we get:

$$R^{(L)} = N \sum_{k=0}^{L-1} \binom{N}{k} p^{N-k} q^k + \sum_{k=L}^{N} k \binom{k-1}{L-1} p^{k-L} q^L \tag{4}$$

The expression in (4) is derived by noting that the procedure will require N polls (i.e., poll all respondents) if $L-1$ or less respondents are active. Otherwise, exactly k respondents are polled if the first $k-1$ polls result in discovering $L-1$ active respondents and the last poll discovers an active respondent.

If the TDMA collection protocol is used, a single solicitation message is sent which reaches all N respondents and

thus we have for all collection objectives y: $C^{(y)}_{TDMA} = 1$ and $S^{(y)}_{TDMA} = N$. The average number of slots needed to achieve the collection objective will be the same as the number of polls required to reach the same collection objective when polling is used. We thus get that for all collection objectives y, $D^{(y)}_{TDMA} = R^{(y)}$.

5 Staged TDMA

5.1 Description

Staged TDMA is a generalization of both the polling and TDMA protocols described above. The set of potential respondents is subdivided into disjoint groups, say g_i for $i = 1, \ldots, M$. Where M is less than or equal to the total number of respondents. Responses are gathered by having the collector send a multicast message to each group one at a time. All the respondents in a group are ordered and they are allocated slots in which to respond if they are active. If the collection objective is achieved after or during the exploration of the ith group, the procedure is terminated. Otherwise, the collector goes on to explore the $(i+1)$th group and so on. The multicast solicitation messages sent by the collector contain the collection objective. Note that, in general, the objective declared in the $(i+1)$th solicitation message is a "reduced" version of the one declared in the ith message. The amount of reduction is determined by the number of responses collected in the ith stage. All respondents solicited in stage i operate with the knowledge of the collection objective and the number of the not-yet-solicited respondents. Thus during a stage a determination can be made when the collection objective has been achieved or cannot be achieved because the number of unexplored respondents is not sufficient.

Here all respondents in group g_i are solicited once they receive the collector's multicast message. Each respondent is subsequently enabled during its allocated slot. (Similar ideas for the staging of a search can be found in [21].)

We distinguish between *fixed-group* and *adaptive-group* staged TDMA. When fixed groups are used, a set of mutually exclusive and collectively exhaustive groups are determined *a priori*. In an adaptive-group staged procedure, on the other hand, we decide on the constitution of a group after the result of exploring the previous groups is known. The use of optimal adaptive groups will intuitively incur less or equal cost than the use of optimal fixed groups. Adaptive groups, however, may require the use of multiple destination addresses in multicast messages, as single multicast addresses cannot be set up ahead of time.

The staged TDMA protocol has the advantage that it may achieve its objective without involving (i.e., soliciting) all the respondents. It may, however, require somewhat more time to complete when compared to the single-stage TDMA protocol described above because of the delay involved in sending solicitation messages. For a performance measure that incorporates the time to complete, as well as the number of involved respondents, the performance of the staged TDMA protocol can be optimized by selecting the groups appropriately. Observe, however, that if the collection objective is to identify all active respondents, as is the case in [6, 8], then no advantage is gained by staging the TDMA collection procedure. In such situations a single-stage TDMA procedure is always superior to polling or to staged TDMA.

5.2 Analysis and Optimization

We first observe that the average number of respondents that need to be enabled in a staged TDMA protocol will be the same as the average number enabled in a (single-stage) TDMA protocol. Thus we have that for all collection objectives y: $D^{(y)}_{st-TDMA} = D^{(y)}_{TDMA} = R^{(y)}$, where $R^{(y)}$ is given in equation (4), for the L or Maximum collection objective. We emphasize that the above is true regardless of the method of staging (fixed or adaptive) or of the actual grouping used. We next consider the other two performance measures.

5.2.1 Fixed Groups

When fixed-group staged TDMA is used a set of M disjoint groups, g_i, for $i = 1, \ldots, M$ are given. The size of group g_i is given by n_i and $\sum_{i=1}^{M} n_i = N$. (We address the determination of the best such grouping shortly.) We let $\underline{n} = (n_1, n_2, \ldots, n_M)$. Both the collector and respondent solicitation costs will be a function of $\underline{n}$. We also define the integer $1 \le J(k) \le M$ as the smallest integer such that $\sum_{i=1}^{J(k)} n_i \ge k$ for $1 \le k \le N$. This represents the number of groups that need to be solicited if k respondents should be enabled.

Using the same reasoning as that used in the derivation of equation (4), we obtain the following set of expressions:

$$\begin{aligned} C^{(L)}(\underline{n}) &= M \sum_{k=0}^{L-1} \binom{N}{k} p^{N-k} q^k \\ &\quad + \sum_{k=L}^{N} J(k) \binom{k-1}{L-1} p^{k-L} q^L \end{aligned} \tag{5}$$

$$\begin{aligned} S^{(L)}(\underline{n}) &= N \sum_{k=0}^{L-1} \binom{N}{k} p^{N-k} q^k \\ &\quad + \sum_{k=L}^{N} \sum_{i=1}^{J(k)} n_i \binom{k-1}{L-1} p^{k-L} q^L \end{aligned} \tag{6}$$

We next investigate how the total cost of the fixed-group, staged TDMA procedure may be optimized by selecting the appropriate fixed group sizes. For any given collection objective y the total cost is given by:

$$\begin{aligned} A^{(y)}(\underline{n}) &= \alpha R^{(y)} + \beta S^{(y)}(\underline{n}) + \gamma C^{(y)}(\underline{n}) \\ &= \alpha R^{(y)} + B^{(y)}(\underline{n}) \end{aligned} \tag{7}$$

Thus minimizing the total cost can be achieved by minimizing $B^{(y)}(\underline{n})$ which satisfies the following recursive equation:

$$\begin{aligned} B^{(\ell)}(\underline{v}) &= \beta v_1 + \gamma \\ &+ \sum_{x=0}^{v_1} B^{(\ell-x)}(\underline{v}^{-1}) \binom{v_1}{x} p^{v_1-x} q^x \end{aligned} \tag{8}$$

where $\underline{v} = (v_1, v_2, \ldots, v_T)$ and $\underline{v}^{-1} = (v_2, v_3, \ldots, v_T)$. The cost in equation (8) is derived as the sum of the solicitation cost for the first group (of size v_1) plus the cost of the collection protocol as it proceeds through the rest of the groups with a diminished collection objective. To complete the expression in (8) we need the following boundary conditions where $\underline{v} = (v_1, v_2, \ldots, v_T)$:

$$B^{(j)}(\underline{v}) = 0 \qquad for \quad j \leq 0 \tag{9}$$

$$B^{(\ell)}(v_1) = \beta v_1 + \gamma \qquad for \quad 0 < \ell \leq v_1 \tag{10}$$

$$B^{(\ell)}(\underline{v}) = \beta \sum_{i=1}^{T} v_i + \gamma T \; for \; \ell > \sum_{i=1}^{T} v_i \tag{11}$$

This last boundary condition warrants some explanation: This objective is equivalent to the objective "find all active" which requires the solicitation and enabling of all the remaining respondents thus the solicitation cost is as given above.

In order to determine the best fixed grouping that will minimize the $B^{(\ell)}(\underline{n})$ for a given q, β and γ, one straightforward method is to enumerate all potential groupings of the N respondents and evaluate the cost of each (using (8) or the appropriate expressions in (5) - (6)). The optimum grouping is the one with the minimum such cost. The approach just described is obviously not feasible as it is prohibitively time consuming even for moderate values of N. We thus adopt a heuristic approach aimed at determining a near-optimal grouping. Our approach is based on the assumption that the optimal solicitation cost for achieving objective y given the set of m respondents are subdivided into T groups, satisfies the following recursive relationship (based on (8)):

$$B^{(\ell)}_{opt}(\underline{v}_{T,m}) = \min_{0<v_1\leq m-T+1} \{\beta v_1 + \gamma + B^{(rnd[\ell - qv_1])}_{opt}(\underline{v}_{T-1,m-v_1})\} \tag{12}$$

where $rnd[\bullet]$ rounds its argument to the nearest integer. Note that if a group of size v_1 is tested, an average of qv_1 responses are anticipated. Thus, the expression in (12) is based on the assumption that the optimal grouping for finding exactly ℓ is approximated by the optimum grouping for finding an average of ℓ.

The optimization procedure is thus as follows:

1. For each possible number of groups $M = 1, \ldots, N$ determine the grouping of the respondents into M groups using (12) and the following boundary conditions

$$B^{(j)}(\underline{v}_{T,m}) = 0 \qquad for \quad j \leq 0 \tag{13}$$

$$B^{(\ell)}(\underline{v}_{1,m}) = \beta m + \gamma \quad for \quad 0 < \ell \leq m \tag{14}$$

$$B^{(\ell)}(\underline{v}_{T,m}) = \beta m + \gamma T \; for \; \ell > m \tag{15}$$

   Note that the values of the boundary conditions above are independent of the grouping used. Thus, whenever, while using (12), the value of B is evaluated using these boundary conditions, we assume that the T remaining groups are such that the last group contains $m - T + 1$ respondents and the other $T - 1$ groups contain one respondent each.

2. Choose the grouping (from among the N different ones produced in Step 1) that yields the lowest cost as evaluated by (8).

We can judge how near-optimal the grouping found using the heuristic above by comparing its cost to the cost of the best adaptive grouping (as determined in the next subsection). This latter cost is a lower bound on the best fixed-group cost.

5.2.2 Adaptive Groups

In contrast to the fixed-groups protocol described above where a given set of M groups are used, the groups used in an adaptive protocol can be described by a tree. The root of the tree represents the group solicited in the first stage. The number of active respondents discovered at the end of the first stage determines the size of the next group to be solicited if the objective has not been reached. The same occurs at the conclusion of the second stage and so on.

Rather than proceeding as before by analyzing the protocol for any given tree of groups, we proceed directly to the discussion of the optimization step. As in the fixed-group staged TDMA case, the only part of the cost that is amenable to optimization is the total solicitation cost. We define $B^{(y)}_{opt}(m)$ as the optimum total solicitation cost when the number of unsolicited respondents is m. Using the same arguments leading to (12) we can write:

$$B^{(\ell)}_{opt}(m) = \min_{0<n\leq m} \{\beta n\gamma + + \sum_{x=0}^{n} B^{(\ell-x)}_{opt}(m-n) \binom{n}{x} p^{n-x} q^x\} \tag{16}$$

The above equation with the following boundary conditions can be used to determine the optimum tree of groups that needs to be used.

$$B^{(j)}_{opt}(m) = 0 \qquad for \quad j \leq 0 \tag{17}$$

$$B^{(j)}_{opt}(0) = 0 \quad ; \quad B^{(1)}_{opt}(1) = \beta + \gamma \tag{18}$$

$$B^{(\ell)}_{opt}(m) = \beta m + \gamma \quad for \quad \ell > m \tag{19}$$

This last boundary condition stems from the fact that, as the objective stated is equivalent to finding all active users, it is best to use a single stage (i.e., a single broadcast solicitation message) which will incur the given solicitation cost.

6 Group Testing

6.1 Description

The group testing response collection procedure is initiated by a broadcast solicitation message sent by the collector and

received by all respondents. Once this message is received by all respondents, the channel operates in the slotted mode where a group of respondents is enabled at the beginning of each slot. The choice of group to enable is determined entirely by the respondents by observing the channel activity and does not require intervention by the collector. The protocol operates in a similar manner to the one described in [6], with the major difference being that the protocol will terminate whenever the collection objective is achieved.

Each respondent observes the channel activity during each slot and updates its knowledge of the state of the respondents accordingly. The state of the system is described by membership in four sets [6, 1]: a *classified set*, a *binomial set*, a *defective set*, and a *conflicted set*. More details on the operation of this protocol, including the definitions of these sets can be found in [6].

6.2 Analysis

The size of the groups enabled can be found through the solution of a set of recursive equations shown below. In the following $H^{(y)}(n)$ denotes the average number of slots needed to satisfy the collection objective y when the binomial set is of size n, $F^{(y)}(m,n)$ denotes the average number of slots when the defective set is of size m and the binomial set is of size $n-m$ and $G^{(y)}(k,m,n)$ denotes the average number of slots when the defective set is of size k, the conflicted set is of size m and the binomial set is of size $n-m$. Then we can write for $\ell \geq 1$: (These equations are similar to those shown in [6]. There are, however, small but critical differences that have to do with the fact that the collection objective is now an influencing parameter.)

$$H^{(\ell)}(n) = 1 + \min_{1\leq x\leq n}\{P_0 H^{(\ell)}(n-x) + P_1 H^{(\ell-1)}(n-x) + (1-P_0-P_1)G^{(\ell)}(0,x,n)\}, \quad n \geq 1 \tag{20}$$

$$F^{(\ell)}(m,n) = 1 + \min\{A_1, A_2\}, \quad n \geq 1 \tag{21}$$

$$A_1 = \min_{1\leq x\leq m}\{P_2 F^{(\ell)}(m-x,n-x) + P_3 H^{(\ell-1)}(n-x) + (1-P_2-P_3)G^{(\ell)}(0,x,n)\}$$

$$A_2 = \min_{m<x\leq n}\{P_4 H^{(\ell-1)}(n-x) + (1-P_4)G^{(\ell)}(m,x,n)\}$$

$$G^{(\ell)}(0,m,n) = 1 + \min_{1\leq x<n}\{P_5 G^{(\ell)}(0,m-x,n-x) + P_6 + F^{(\ell-1)}(m-x,n-x) + (1-P_5-P_6)G^{(\ell)}(0,x,n)\}, \quad n \geq m \geq 2 \tag{22}$$

$$G^{(\ell)}(k,m,n) = 1 + \min\{B_1, B_2\}, \quad n \geq m \geq 2, \quad m-1 \geq k \geq 1 \tag{23}$$

$$B_1 = \min_{1\leq x\leq k}\{P_7 G^{(\ell)}(k-x,m-x,n-x) + P_8 F^{(\ell-1)}(m-x,n-x) + (1-P_7-P_8)G^{(\ell)}(0,x,n)\}$$

$$B_2 = \min_{k<x\leq m}\{P_9 F^{(\ell-1)}(m-x,n-x) + (1-P_9)G^{(\ell)}(k,x,n)\}$$

where P_0 = probability that no transmission occurs $= q^x$ and P_1 = probability that exactly one transmission occurs $= xq^{x-1}p$. The expressions for P_2 through P_9 are identical to those in [6, equations (5.1)-(5.4)] and are not repeated here.

In addition, the following boundary conditions are applicable:

$$H^{(0)}(n) = H^{(\ell)}(0) = 0; \quad F^{(0)}(m,n) = 0 \tag{24}$$

$$F^{(\ell)}(0,n) = H^{(\ell)}(n) \tag{25}$$

$$F^{(\ell)}(1,n) = 1 + H^{(\ell-1)}(n-1) \tag{26}$$

$$G^{(0)}(k,m,n) = G^{(\ell)}(0,1,n) = 0 \tag{27}$$

$$G^{(\ell)}(1,m,n) = 1 + F^{(\ell-1)}(m-1,n-1) \tag{28}$$

$$G^{(\ell)}(k,2,n) = 2 + H^{(\ell-2)}(n-2) \tag{29}$$

Since all respondents are initially in the binomial set we have that the average number of slots needed to achieve the collection objective is $D_{GT}^{(y)} = H^{(y)}(N)$. If the group testing collection protocol is used, a single solicitation message is sent which reaches all N respondents and thus we have for all collection objectives y: $C_{GT}^{(y)} = 1$ and $S_{GT}^{(y)} = N$.

7 Staged Group Testing

7.1 Description

Staged group testing is a generalization of both the polling and (the single stage) group testing protocols. The protocol operates in a similar manner to the staged TDMA protocol. The set of respondents are solicited in a set of stages, where each stage begins by a multicast solicitation message sent to a subset of the respondents. As before these solicitation messages contain the collection objective and are modified from the initial objective as responses are collected in each stage. The groups of respondents solicited in different stages are disjoint.

Whereas in the staged TDMA protocol solicited respondents are enabled during slots allocated to each individually, in the staged group testing protocol solicited respondents are enabled according to a group testing procedure that involves only the members of the group being explored during the current stage. Within stage i a group testing procedure that involves only the respondents in group g_i is carried out. At the beginning of each slot in stage i the respondents involved know the distribution of the respondents in g_i into each of the classified, binomial, defective and conflicted sets. In addition they know the collection objective (declared by the collector in its ith solicitation message) and the number of remaining and not-yet-solicited respondents. This information allows the procedure to terminate before all the respondents in g_i have been enabled

when either the collection objective is achieved or it is determined that the objective is not attainable as the number of remaining respondents in this and further stages is not sufficient.

We again distinguish between fixed and adaptive groups as was done for the staged TDMA protocol. By appropriately selecting the groups used in either a fixed or adaptive group staged group testing procedure, its cost may be minimized.

7.2 Analysis and Optimization

7.2.1 Fixed Groups

As before we are given a set of M disjoint groups defined by the vector $\underline{n}$ whose elements are the sizes of each group. We first observe that for a given value of $\underline{n}$, the respondent solicitation cost (i.e., the number of respondents solicited) and the collector solicitation cost (i.e., the number of solicitation message sent by the collector) will be identical to those obtained for the fixed groups staged TDMA protocol in section 5.2.1. We thus assert that for any collection objective y:

$$C^{(y)}_{FG,st-GT}(\underline{n}) = C^{(y)}_{FG,st-TDMA}(\underline{n}) \quad and \tag{30}$$

$$S^{(y)}_{FG,st-GT}(\underline{n}) = S^{(y)}_{FG,st-TDMA}(\underline{n}) \tag{31}$$

Equations (5) - (6) contain the appropriate expressions for the L or Maximum collection objective.

It remains to determine the average delay cost (or the average number of response slots required for the collection objective to be satisfied). We first note that unlike the staged TDMA protocol where the average delay cost was the same as that incurred by the single stage TDMA protocol, here staging will, in general, affect the delay cost. In order to capture the fact that the group testing procedure within any one stage operates with the knowledge of the number of the not-yet-solicited respondents, we define: $\mathcal{H}^{(y)}(n;t)$ as the average number of slots remaining in a stage when the binomial set is of size n, the number of not-yet-solicited respondents is t and the current collection objective is y. $\mathcal{F}^{(y)}(m,n;t)$ and $\mathcal{G}^{(y)}(k,m,n;t)$ are defined in a similar manner. The above three quantities are related in exactly the same way as the corresponding quantities in section 6.2. The boundary conditions for these quantities are essentially the same as those shown in section 6.2.

We note the following equivalence relations:

$$\mathcal{H}^{(y)}(n;0) = H^{(y)}(n); \quad \mathcal{H}^{(L)}(n;t) = H^{(L)}(n) \tag{32}$$

The average delay cost satisfies the following recursive relationship:

$$D^{(\ell)}(\underline{v}) = \mathcal{H}^{(\ell)}(v_1; \sum_{i=2}^{T} v_i) + \sum_{x=0}^{v_1} D^{(\ell-x)}(\underline{v}^{-1}) \binom{v_1}{x} p^{v_1-x} q^x \tag{33}$$

where $\underline{v}$ and $\underline{v}^{-1}$ are as defined in section 5.2.1.

In addition we use the following boundary conditions

$$D^{(j)}(\underline{v}) = 0 \qquad for \quad j \le 0 \tag{34}$$

$$D^{(\ell)}(v_1) = H^{(\ell)}(v_1) \quad for \quad 0 < \ell \le v_1 \tag{35}$$

$$D^{(\ell)}(\underline{v}) = \sum_{i=1}^{T} H^{(v_i)}(v_i) \ for \ \ell > \sum_{i=1}^{T} v_i \tag{36}$$

For a given set of weights, α, β, and γ, a near-optimum set of fixed groups can be determined by assuming the following recursive equation for the optimal total cost, $A^{(y)}_{opt}(\underline{v}_{T,m})$ for achieving objective y given a set of m respondents are subdivided into T groups. (see equation (12) and arguments leading to it):

$$A^{(\ell)}_{opt}(\underline{v}_{T,m}) = \min_{0<v_1\le m-T+1} \left\{ \alpha \mathcal{H}^{(\ell)}(v_1; m-v_1) + \beta v_1 + \gamma + A^{(rnd[\ell-qv_1])}_{opt}(\underline{v}_{T-1,m-v_1}) \right\} \tag{37}$$

where the following boundary conditions are obeyed:

$$A^{(j)}_{opt}(\underline{v}_{T,m}) = 0 \qquad for \quad j \le 0 \tag{38}$$

$$A^{(\ell)}_{opt}(\underline{v}_{T,m}) = A^{(m)}_{opt}(\underline{v}_{T,m}) \ for \ \ell > m \tag{39}$$

$$A^{(\ell)}_{opt}(\underline{v}_{1,m}) = \alpha H^{(\ell)}(m) + \beta m + \gamma \ for \ 0 < \ell \le m \tag{40}$$

The optimization procedure using (37) is described in section 5.2.1. Also, as for the staged TDMA case, the adaptive groups optimum cost derived in the next subsection can be used as a lower bound by which we can judge the "goodness" of our heuristically obtained fixed grouping.

7.2.2 Adaptive Groups

The optimum groups to use in an adaptive-group, staged group testing procedure can be determined by considering the following recursive equation (see equation (16) and arguments leading to it):

$$A^{(\ell)}_{opt}(m) = \min_{0<n\le m} \left\{ \alpha \mathcal{H}^{(\ell)}(n; m-n) + \beta n + \gamma + \sum_{x=0}^{n} A^{(\ell-x)}_{opt}(m-n) \binom{n}{x} p^{n-x} q^x \right\} \tag{41}$$

where the following boundary conditions are satisfied:

$$A^{(j)}_{opt}(m) = 0 \qquad for \quad j \le 0 \tag{42}$$

$$A^{(j)}_{opt}(0) = 0 \quad ; \quad A^{(1)}_{opt}(1) = \alpha + \beta + \gamma \tag{43}$$

$$A^{(\ell)}_{opt}(m) = \alpha H^{(m)}(m) + \beta m + \gamma \ for \ \ell > m \tag{44}$$

8 Numerical Examples

8.1 Polling and TDMA

First we consider the performance of polling and TDMA protocols. The quantity of interest for both protocols is $R^{(\nu)}$ (see section 4.2). For the polling protocol, this quantity indicates the average number of polling messages sent (which is the same as the number of response slots required and the number of respondents enabled). For the TDMA protocol, the quantity indicates the number of response slots required. We refer to this measure generically as the average number of "steps". The variation of the average number of steps is shown as a function of q for the L or Maximum collection objective in Figure 1. Systems where the number of respondents N is 20 are considered. (Recall that for the TDMA protocol, exactly one solicitation message is sent and that all N respondents are solicited regardless of the collection objective and of the value of q.)

For the L or Maximum collection objective (Figure 1) as the likelihood of receiving a response from a respondent increases, the average number of steps required decreases. The average number of steps increases if the value of L required to achieve the collection objective increases.

In general, polling will be preferred over TDMA only if the weight of the respondent solicitation cost, β, is high as polling's only advantage is that less respondents are solicited.

8.2 Staged TDMA

The delay cost of the staged TDMA procedure is the same as that for a TDMA protocol. The variations of this cost with q are shown in Figure 1. As mentioned earlier, the only advantage of a staged TDMA protocol is that it may incur a lower respondent solicitation cost at the expense of a slightly higher collector solicitation cost. As an example, we consider achieving the L or Maximum collection objective using a fixed group staged TDMA procedure in a system where the 20 respondents are subdivided into four groups of sizes (6, 6, 4, 4). The respondent and collector solicitation costs are shown as a function of q in Figures 2 and 3, respectively. Whereas in the single stage TDMA protocol the respondent solicitation cost is always 20, using a staged procedure can provide for a lower cost especially for high values of q. This is achieved at the expense of increasing the collector solicitation cost from the value 1 in the single stage TDMA protocol to a value between 1 and 4 (as shown in Figure 3) in this case.

Table 1 shows near-optimal groupings of 15 respondents in a fixed-group procedure for various values of the cost weights and for the L or Maximum collection objective. The table also shows the cost of using an optimal adaptive-group, staged TDMA procedure. Observe that the heuristically obtained fixed groupings result in the same or slightly higher costs than the optimal adaptive groupings. The equality in cost happens when the optimal adaptive grouping is equivalent to the fixed grouping shown. Note also that the "best" fixed-group staged TDMA procedure is sometimes one where there is one group of size 15 or 15 groups of size one. This matches our intuition that in certain instances, a single stage TDMA procedure or polling will be best.

8.3 Group Testing

Group testing (in a single stage) incurs a respondent and collector solicitation costs of N and 1, respectively regardless of the value of q. Figure 4 shows the average delay cost as a function of q for the L or Maximum collection objective. By comparing to Figure 1, we make the following observations: 1)For the entire range of q the use of group testing provides for a lower or equal delay cost than a single stage TDMA protocol. 2)The group testing collection procedure adapts to the (single stage) TDMA procedure (i.e., in each slot a group of size 1 is enabled) when $q \geq \frac{1}{\sqrt{2}}$. (This was found to be true in all the numerical experiments we conducted. No formal proof is available yet.)

8.4 Staged Group Testing

The respondent and collector solicitation costs of the staged group testing procedure are the same as those for the staged TDMA procedure. For the L or Maximum collection objective the variations of these costs with q are shown in Figures 2 and 3, where the respondents are subdivided into four groups of sizes (6,6,4,4). The delay cost of the staged group testing procedure for the same grouping of the respondents is shown in Figure 5. Comparing this delay with that incurred by a single stage group testing procedure (as shown in Figure 4) we observe that: 1)For values of $q \geq \frac{1}{\sqrt{2}}$ the delay cost for the two approaches is the same since in both the group testing procedure adapts to a TDMA procedure. 2) For values of $q < \frac{1}{\sqrt{2}}$, the delay incurred is lower when single stage group testing is used. Particularly, in the limit as q approaches zero, the single stage group testing procedure needs only one group test to determine that the respondents are not active, whereas the staged group testing procedure needs a number of group tests equal to the number of groups of respondents.

Table 2 shows the grouping of 15 respondents in a fixed-group, staged group testing procedure for various values of the cost weights and for the L or Maximum collection objective. It also shows the cost when an optimal adaptive-group, staged group testing procedure is used. Observe the following: 1)As in the staged TDMA case, the near-optimal fixed groupings achieve the same or slightly higher cost than the optimal adaptive groupings. 2)The costs of the staged group testing procedure are close to those of the staged TDMA procedure if the parameters are such that the optimal grouping results in small size groups. Otherwise, the staged group testing costs can be lower.

9 Concluding Remarks

In this paper we have considered response collection strategies that can be used over a multiple access channel. We were motivated by some distributed computing applications to define a set of collection objectives. Five protocols that can be used to achieve these collection objectives were investigated: Polling, TDMA, staged TDMA, group testing and staged group testing. In analyzing the performance of these protocols, three cost components were taken into account: the number of steps required to complete the objective, the number of solicitations required by the collector, and the the number of respondents receiving solicitation messages. The idea of staging stems from the inclusion of

the latter two cost components and from the fact that the request procedure will terminate once the collection objective has been achieved.

Our findings are summarized in Table 3 where we use the terms low, medium, and high to denote relative values of the costs. Our conclusion is that, in general, a suitably optimized adaptive-group, staged group testing protocol can achieve the best performance. A near optimal fixed-group staged group testing procedure can achieve almost similar performance but can be easier to implement as the groups are determined a priori.

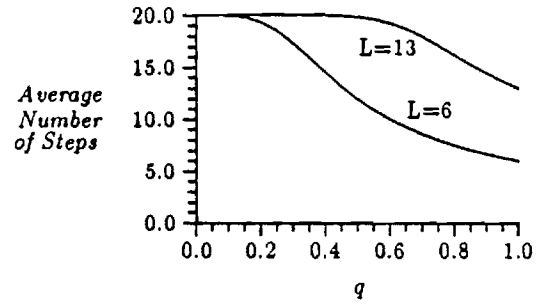


Figure 1: Average Number of Steps for the L or Maximum Collection Objectives (Polling or TDMA)

References

[1] M. Sobel and P. A. Groll, "Group testing to eliminate efficiently all defectives in a binomial sample," *The Bell Systems Technical Journal*, September 1959.

[2] F. K. Hwang, "On finding a single defective uisng binomial group testing," *Journal of the American Statistical Association*, vol. 69, pp. 146–150, MARCH 1974.

[3] M. R. Garey and F. K. Hwang, "Isolating a single defective using group testing," *Journal of the American Statistical Association*, vol. 69, pp. 151–153, March 1974.

[4] M. C. Hu, F. K. Hwang, and J. K. Wang, "A boundary problem for group testing," *SIAM Journal of Algebra and Discrete Mathematics*, vol. 2, pp. 81–87, June 1981.

[5] J. Wolf, "Born again group testing: Multiaccess communications," *IEEE Transactions on Information Theory*, vol. 31, pp. 185–191, March 1985.

[6] T. Berger, N. Mehravari, D. Towsley, and J. Wolf, "Random multiple-access communication and group testing," *IEEE Transactions on Communications*, vol. 32, pp. 769–779, July 1984.

[7] N. K. Garg and S. Mohan, "Group testing with capture for random access communication," *IEEE Transactions on Communications*, vol. 35, pp. 849–854, August 1987.

[8] D. Kurtz and M. Sidi, "Multiple-access algorithms via group tetsting for heterogeneous population of users," *IEEE Transactions on Communications*, vol. 36, pp. 1316–1323, December 1988.

[9] T. K. Sellis, "Multiple-query optimization," *ACM Transactions on Database Systems*, vol. 13, pp. 23–52, MARCH 1988.

[10] M. H. Ammar, "Teletext-like information delivery using broadcast polling," *Computer Networks and ISDN Systems*, vol. 12, pp. 107–115, MARCH 1987.

[11] M. H. Ammar and H. J. Kim, "Prototyping a broadcast delivery information system," Tech. Rep. GIT-ICS-90/16, Georgia Institute of Technology, Atlanta, GA, 1990.

[12] G. Herman, G. Gopal, K. Lee, and A. Weinrib, "The datacycle architecture for very high throughput databases," in *Proceedings of SIGMOD*, ACM, 1987.

[13] J. W. Wong and M. H. Ammar, "Analysis of broadcast delivery in a videotex system," *IEEE Transactions on Computer*, vol. 34, pp. 863–866, September 1985.

[14] L. Lamport, "The implementation of reliable distributed multiprocess systems," *Computer Networks*, vol. 2, pp. 95–114, 1978.

[15] D. Gifford, "Weighted voting for replicated data," in *Proceedings of 7th Symposium on Operating Systems*, pp. 150–162, ACM, 1979.

[16] M. Ahamad and M. H. Ammar, "Performance characterization of quorum-consensus algorithms for replicated data," *IEEE Transactions on Software Engineering*, vol. 15, pp. 492–496, April 1989.

[17] D. Barbara and H. Garcia-Molina, "Mutual exclusion in partitioned distributed systems," *Distributed Computing*, vol. 1, pp. 119–132, 1986.

[18] S. Y. Cheung, M. Ahamad, and M. H. Ammar, "Optimizing vote and quorum assignments for reading and writing replicated data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 1, pp. 387–397, September 1989.

[19] S. Mullender and P. Vitanyi, "Distributed matchmaking," *Algorithmica*, vol. 3, 1988.

[20] M. H. Ammar and G. N. Rouskas, "On the performance of protocols for collecting responses over a multiple access channel," Tech. Rep. GIT-ICS-90/53, Georgia Institute of Technology, Atlanta, GA, 1990.

[21] J. Bernabeu, M. H. Ammar, and M. Ahamad, "Optimal selection of multcast groups for resource location in a distributed system," in *Proceedings of INFOCOM '89*, IEEE, 1989.

Cost Measure	Polling	TDMA	Staged TDMA	Group Testing	Staged Group Testing
Delay Cost (no. of slots)	High	High (=Polling)	High (=Polling)	Low	Medium
Respondent Solicitation Cost	Low	High (= N)	Medium	High (= N)	Medium (= staged TDMA, for same grouping)
Collector Solicitation Cost	High	Low (=1)	Medium	Low (=1)	Medium (= staged TDMA, for same grouping)

Table 3: Relative Performance of the Various Protocols

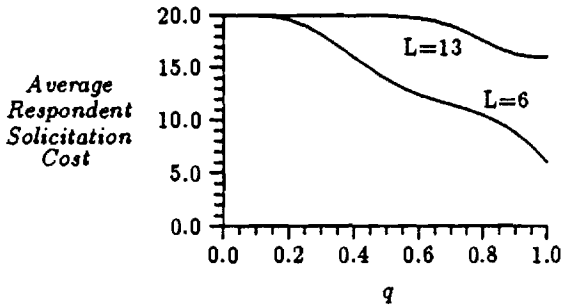


Figure 2: Average Respondent Solicitation Cost for the L or Maximum Collection Objectives (Fixed Group Staged-TDMA)

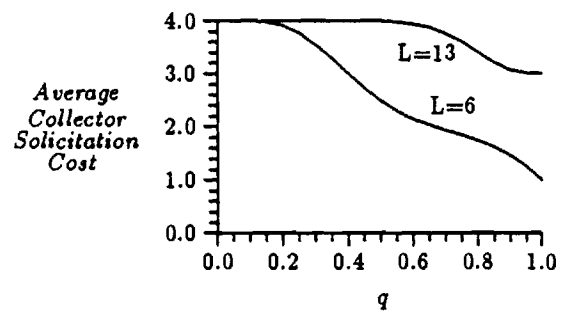


Figure 3: Average Collector Solicitation Cost for the L or Maximum Collection Objectives (Fixed Group Staged-TDMA)

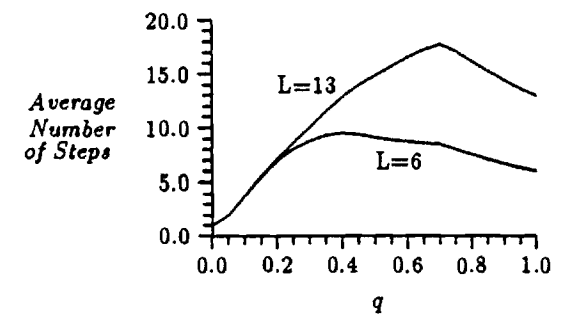


Figure 4: Average Delay Cost for the L or Maximum Collection Objectives (Group Testing)

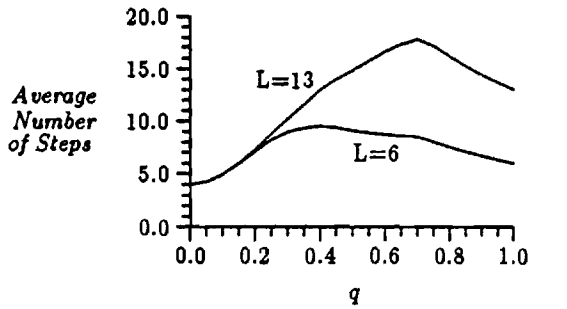


Figure 5: Average Delay Cost for the L or Maximum Collection Objectives (Fixed Group Staged Group Testing)

L	q	α	β	γ	Fixed Groups	FG cost	AG cost
3	.85	1	10	1	3 1 ... 1	40.35	40.29
1	.9	1	10	1	1 ... 1	13.33	13.33
1	.1	1	10	1	1 ... 1	95.29	95.29
1	.1	1	.1	1	15	10.44	10.44
8	.8	1	.1	1	10 3 1 1	12.45	12.34
8	.1	1	10	1	15	166.00	166.00

Table 1: Near-Optimal Fixed Group and Optimal Adaptive Group Staged TDMA (L or Maximum)

L	q	α	β	γ	Fixed Groups	FG cost	AG cost
3	.85	1	10	1	3 1 ... 1	40.35	40.29
1	.9	1	10	1	1 ... 1	13.33	13.33
1	.1	1	10	1	2 2 2 1 ... 1	93.35	92.09
1	.1	1	.1	1	15	4.33	4.33
8	.8	1	.1	0	10 3 1 1	12.45	12.28
8	.1	1	10	1	15	153.86	153.86

Table 2: Near-Optimal Fixed Group and Optimal Adaptive Group Staged Group Testing (L or Maximum)

Paper #9

Resource Finding in Store–and–Forward Networks*

José M. Bernabéu-Aubán *Mustaque Ahamad* *Mostafa H. Ammar*
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332

Abstract

We model the process of searching for a resource in a distributed system whose nodes are connected through a store–and–forward network. Based on this model, we show a lower bound on the number of messages needed for finding a resource when nothing is known about its location. The model also helps us establish some results about the complexity of finding optimal algorithms to locate a resource when the probability distribution for the location of the resource is known. We show that the optimization problem is NP-hard for general networks. Finally we develop an algorithm for tree networks which can be specialized to polynomial algorithms for a class of trees. (The polynomial algorithms can be used as the basis of heuristic algorithms for general networks.) An application of this algorithm for path networks can be adapted to find optimal search algorithms for bidirectional ring networks.

1 Introduction

Distributed systems need to implement algorithms for finding the location of remote resources to reduce the complexity of their use. We investigate the communication cost of location finding algorithms in a store–and–forward network. We consider two situations. In the first, our goal is to investigate resource finding algorithms when a searcher node does not know where information about the resource resides. In the second, we assume the searcher node has some statistical information (e.g., a probability distribution). Such situations can arise in a distributed system with the commonly used schemes such as name servers [1] or hint tables [2]. For example, when the name server node fails, the algorithm used by the node that wants to find the resource location (searcher node) must work without exact knowledge about the nodes that are likely to know the resource location.

When the searcher has no information about the location of a resource (or of its references), we show in this paper:
1) The expected number of messages used in searching for the resource has a lower bound of $\Theta(\sqrt{\frac{\mu}{\lambda}N})$, where N is the number of nodes in the network, μ is the rate at which the resource moves and λ is the rate at which requests for the resource arrive. This lower bound includes the messages needed to update the references to the resource. Furthermore, there is a nondeterministic algorithm reaching this lower bound in a completely connected network. 2) For arbitrary networks, the expected number of messages needed to find the location of a resource that has n references in the network has an upper bound of $N - n$ (this bound is tight).

When the searcher has a probability distribution describing the likelihood of a particular node knowing the location of a resource, we show: 1) The problem of determining the optimal way of searching an arbitrary network to minimize the expected number of messages used is NP-hard. 2) For complete networks, in which the cost of sending a message through a link is the same for all links, the problem of selecting the optimal way to search the network is shown to be equivalent to sorting. 3) An algorithm is developed for tree networks which improves on exhaustive search. We show a polynomial algorithm to find the optimal way to search for a resource in a bidirectional ring network.

The resource finding problem has been addressed by several researchers. In [3] and [4], methods suitable for store–and–forward networks are presented and it is shown that their average cost when the ratio of resource request and movement rate is a constant, is $O(\sqrt{N})$ in complete networks. However, these methods require that additional information be used by each node. For example, two of the forwarding protocols studied in [5] require that all nodes store an address for each of the resources. Similarly, in [3], each node must have two sets of nodes associated with it.

The problem of searching has also been addressed in the literature in a different context [6,7]. However, the solutions obtained are not applicable to location finding in distributed systems. There exist other schemes which are useful in a particular type of network [8,9]. These schemes

*This work was supported in part by NSF grants CCR–8806358 and NCR–8604850.

CH2826-5/90/0000/0819/$01.00 © 1990 IEEE

may not be efficient if used in a store-and-forward network.

In section 2 we introduce our model of the system and the strategy used for finding a resource which is called *serial search*. In section 3 we show a worst case analysis of serial search when the searcher has no knowledge about the location of the resource. In section 4 we study the problem of finding an optimal way to traverse the system when the searcher has the probability distribution that the resource be at a certain node in the network. The paper is concluded in section 5.

Proofs for the results in this paper may be found in [10].

2 Model

We model the system as a collection of N nodes connected by a store-and-forward communications network. Thus we can represent the system by a graph $G(V, E)$ in which the vertices represent the nodes and the edges represent the communication links in the network. A node's CPU is responsible for both switching arriving messages to appropriate outgoing links and for resource table look up, when a location finding message is received. With each edge e we will associate a cost $c(e)$, which is incurred every time a message traverses the edge. This cost is intended to represent the bandwidth cost associated with traversing the corresponding link. A distributed system will be represented by $G(V, E, c)$.

Each resource in the system resides in one of the nodes, and for each resource, n different nodes store a reference to the resource's current location. The node where the resource resides contains one of the references. The set of nodes containing references to a given resource is called the *well-informed set* of the resource. No other node outside of the well-informed set of a resource has any knowledge about where the resource or any of its references are. We will assume that the references are instantaneously updated when a resource moves, which happens at rate μ. We will further assume that the location of the resource is requested with rate λ. For each $A \subset V$ such that $|A| = n$, we will denote by $Q(A)$ the probability that A is the well-informed set of the resource when its location is requested at one of the nodes not in its well-informed set. Another assumption we will make about the system is that the resource and its references do not move while a search is in progress. We expect that in a real system this will be true for most requests when the resource movement rate is much lower than the resource request rate.

To locate a resource, the searcher follows the links of the network from one node to another until a reference to the requested resource is found. We assume that the search terminates at that point and thus will not consider the cost of propagating the information back to the requesting node. To reduce the time needed to find the location of a resource, it would be better to start several searchers at the same time looking into different nodes. Multiple agent searches have the added complexity of synchronization between the searchers, e.g., if one searcher finds a reference then others need to be informed in order to halt their searches. To avoid such complexity, we only investigate those searches in which only one agent is active at any given time, and we will call them *serial searches*. The results obtained here will inevitably form the basis of an understanding of multiple agent searches. In the next section we describe more formally how the search is carried out.

2.1 Search Model

We can visualize the search process as one in which a searcher agent starts at a node and not finding a reference to the resource, decides on the node to be visited next. It then traverses the link leading to the chosen node in order to search there for the resource reference. This process is repeated until a node storing a reference to the resource is found, at which point the search terminates. Thus, a location finding algorithm can be seen as a rule which selects the next node to be consulted based on the past history of the search.

The only information the searcher can get from consulting a node not in the well-informed set is that the node does not have a resource reference. Due to this reason, the sequence of nodes to be consulted can be laid out statically from the beginning. This is captured in the following definitions.

Definition 1 *Let $G(V, E, c)$ be a graph, and let $s = (v_0, \ldots, v_l)$ be a sequence of nodes in V. We will say that v_p is a first visit if for all k, $k < p$, $v_k \neq v_p$.*

Definition 2 *A walk in a graph $G(V, E, c)$ is a sequence of nodes in V, $(v_0, v_1, \ldots, v_l)$, such that $\{v_i, v_{i+1}\} \in E$ for all $i < l$. The set of walks in G starting at node v is denoted by $\mathcal{W}(G, v)$, and the set of all walks in G is denoted by $\mathcal{W}(G)$.*

To find a resource in a distributed system, a walk $w \in \mathcal{W}(G)$, $w = (v_0, v_1, \ldots, v_l)$, is selected. Then the nodes in the walk are consulted, starting with v_1. The edge $e = \{v_{i-1}, v_i\}$, is traversed only if no reference is found in any of the nodes v_j for $j < i$. This traversal is performed by sending a message from node v_{i-1} to node v_i, thus incurring cost $c(e)$. Node v_i in the walk will consult its local resource table if edge $\{v_{i-1}, v_i\}$ is traversed and v_i is a first visit. If node v_i is not a first visit, that means that the node has already searched its local resource table for the resource without success, and thus there is no need to do so again. In this case, all v_i does is to forward the message to v_{i+1}. In this paper we will consider only the cost incurred by traversing the network links. We will say that the resource has been found, when a node containing a reference to the resource has been reached. The cost incurred by a particular search will then be the sum of the

costs of the links that were traversed before the resource was found.

From the above definition, a walk does not have to visit all nodes in the system. However, the searcher may not succeed in finding the resource if there is a possibility that the well–informed set is a subset of the nodes not included in the walk. If the walk includes all the nodes in the system then it is guaranteed that the resource will be found.

Definition 3 *Given $G(V, E, c)$, a walk $w = (v_0, \ldots, v_l)$ is complete if $V(w) = V$.*

2.2 Serial Traversals

We first formalize the cost of a walk. This cost will be calculated with respect to some probability distribution for the well-informed set, and will represent the expected cost incurred when the search is conducted by following the walk. We first introduce some auxiliary definitions.

Definition 4 *Let $G(V, E, c)$ be a graph, where $c : E \mapsto Z^+$, then we define the length of a walk $w = (v_0, \ldots, v_l) \in \mathcal{W}(G)$, by*

$$\ell(w) = \sum_{i=0}^{l-1} c(\{v_i, v_{i+1}\})$$

The above is the standard definition of the length of a walk in a graph, and is the link cost that the traversal would incur if the resource was not found in the nodes visited by the traversal or if it was found at the last node.

Definition 5 *Let $G(V, E, c)$ be a graph. Let $w = (v_0, \ldots, v_l) \in \mathcal{W}(G, v_0)$, we define the subwalk of w to v, $v \in V$, denoted by w_v, as follows,*

$$w_v = \begin{cases} w & \text{if } v \notin V(w) \\ (v_0, \ldots, v_m) & \text{if } v_m = v \text{ and } v_m \text{ is a first visit} \end{cases}$$

Thus w_v is the shortest (in number of edges) subtraversal containing v, or w itself if it does not contain v.

We will define the cost of a walk based on the probability distribution Q, and it is intended to represent the expected link cost incurred when the searcher uses the plan indicated by a walk of the graph to search for the resource. Given a particular walk w, we will use Q_w to denote the probability that at least one of the nodes visited by w has a reference. We will also use $Q_w(v)$ to denote the probability that when using w as a search path, v be the first node along the search path containing a reference to the resource. Q_w can be obtained from Q in the following way,

$$Q_w = \sum_{A \cap V(w) \neq \phi} Q(A)$$

$Q_w(v)$ can also be obtained from Q by means of the following expression,

$$Q_w(v) = \sum_{A \cap V(w_v) = \{v\}} Q(A)$$

Definition 6 *Let $G(V, E, c)$ be a distributed system, and let Q, $Q : 2^V \mapsto [0, 1]$, be the probability distribution of the well-informed set on 2^V. Then we define the cost of w with respect to Q, denoted as C_w^Q, as*

$$C_w^Q = \sum_{v \in V(w)} \ell(w_v) Q_w(v) + \ell(w)(1 - Q_w)$$

To find a reference to the resource by incurring the least cost, the procedure used by the searcher to decide on the next node to consult should have the following natural properties.

1. Once a node has been consulted, it should not be consulted again.
2. Once the searcher decides to search a new node, it will get to it through a shortest path, and all nodes along the path will have been already searched (otherwise the next node to search would be one of them).

   As a consequence, no shortest path from the current node to the next node selected can contain a not–yet–consulted node.

We can formalize the above in the following lemma,

Lemma 1 *If $w = (v_0, v_1, \ldots, v_l)$, $l \geq 2$, is such that it violates one of the following conditions,*

(a) For all i, j, s.t. $i < j$, if none of the v_m, for $i < m < j$ is first visited, the walk $(v_i, v_i + 1, \ldots, v_j)$ is a shortest path between v_i and v_j.

(b) v_l is a first visit.

Then, there is a $w' \in \mathcal{W}(G, v_0)$, such that $V(w) \subseteq V(w')$ and $C_{w'}^Q \leq C_w^Q$ and $\ell(w') < \ell(w)$.

To facilitate the discussion, we will consider only walks satisfying conditions (a) and (b) of the above lemma.

Definition 7 *A serial traversal in $G(V, E, c)$ is a walk $s = (v_0, v_1, \ldots, v_l)$, satisfying properties (a) and (b) in lemma 1. The set of traversals starting at node v will be denoted by $S(G, v)$. $S(G)$ will denote the set of serial traversals, starting at any node in the graph. The set of complete serial traversals of a graph starting at node $v \in V$ will be denoted by $C(G, v)$, and $C(G)$ denotes the set of complete traversals starting at any node in the graph.*

2.3 Location Finding Algorithms

A location finding algorithm is a "scheduler" which, given a probability distribution for the well-informed set of a resource, produces a serial traversal. In general for each particular starting node, v, a random complete serial traversal may be chosen based on a probability distribution $R_v : \mathcal{C}(G, v) \mapsto [0, 1]$. Thus we will identify a network-wide location finding algorithm with the family of distributions $R = \{R_v\}_{v \in V}$. $R_v(s)$ is the probability that the location algorithm produces serial traversal $s \in \mathcal{C}(G, v)$, when starting the search at node v.

Definition 8 *We define the cost of a location algorithm R starting at node v, given the distribution of the well-informed sets, Q, and denoted by $J(R, Q, v)$, as follows,*

$$J(R, Q, v) = \sum_{s \in \mathcal{C}(G,v)} C_s^Q R_v(s)$$

Thus the cost of the algorithm will be the average cost of the serial traversals it chooses to perform the search. A *deterministic* algorithm is a special case and will always choose the same serial traversal. That is, the distributions R_v will have value 1 for a particular complete traversal and zero for all others.

3 Non-Informed Search

In a distributed system, a node may not have any information about the current location of a remote resource. This could happen either when the node does not monitor information about all remote resources or the information is allowed to become incorrect. The latter is possible because algorithms to maintain correct information about the location of all resources on each of the nodes may not be feasible due to excessive storage requirements and the communication overhead. In this section we will consider the location finding problem when the searcher knows only the number of references, n. Since the searcher does not know Q, the searcher assumes a uniform distribution (the unknown distribution may be different from uniform), i.e., $Q(A) = \frac{1}{\binom{N-1}{n}}$, $\forall A \subset V$, s.t. $|A| = n$. In such a system, we are interested in worst case estimates of the average cost incurred by location finding algorithms.

3.1 Complete Networks

We study the problem for complete networks (in which there is a communication link between every pair of nodes) when $c(e) = 1$ for all $e \in E$. We will use K_N to denote a complete network with N nodes. Thus our cost measure actually accounts for the number of messages used to locate the resource. The cost of an optimal complete traversal in a complete network will constitute a lower bound on the cost of a complete traversal in any of its subnetworks with the same number of nodes. This is so because the set of complete traversals in a subnetwork is a subset of the set of complete walks in the complete network.

In a complete network, any node can be reached directly from any other node. This implies that no serial traversal will ever visit the same node twice. In other words, $\mathcal{C}(G, v)$ is just the set of all $(N-1)!$ permutations of V in which v is the first element.

For an uninformed searcher, any location algorithm will make its selection of the serial traversal independent of the resource being searched. It will only depend on the network and the node starting the search. Our first result states that when Q is uniform, all complete traversals have the same cost.

Lemma 2 *For $G(V, E) = K_N$, and $c(e) = 1$ for all $e \in E$, and for Q the uniform distribution over the sets of n elements, for any $s \in \mathcal{C}(G)$, we have,*

$$C_s^Q = \frac{N}{n+1}$$

A straightforward corollary is that $J(R, Q, v)$ is independent of R when Q is uniform. Thus, when the uninformed searcher assumes Q to be uniform, from its point of view, all complete serial traversals will have the same cost. The searcher could decide to use an algorithm R, for which $R_v(s) = 1$ for a particular traversal, $s \in \mathcal{C}(G, v)$. Thus s would be used each time the location of a resource needs to be found at node v. However, since Q will in general be non-uniform, the *actual* cost the searcher will incur may not be the one given by lemma 2 when R is used. For instance, let $G = K_5, n = 1$ and let $s = (v_0, v_1, v_2, v_3, v_4)$ be such that $R_{v_0}(s) = 1$. If $Q(\{v_4\}) = 1$, then $C(R, Q, v_0) = 4$ instead of 2.5 as given by lemma 2.

The following Theorem establishes the existence of an algorithm (R), whose cost will be exactly the one the searcher expects, regardless of the actual probability distribution.

Theorem 1 *For $G(V, E) = K_N$ and $c(e) = 1$ for all $e \in E$, the location cost when R_v is uniform; i.e., $R_v(s) = \frac{1}{|\mathcal{C}(G,v)|}$ $\forall s \in \mathcal{C}(G, v)$, is independent of Q and is given by*

$$J(R, Q, v) = \frac{N}{n+1}$$

If Q is not uniform, an algorithm which does not choose each traversal with the same probability may have an expected search cost larger than the one given above. Thus, to make sure that the cost of locating a resource is $\frac{N}{n+1}$, R should be uniform ($R_v(s) = R_v(s')$ for all $s, s' \in \mathcal{C}(G, v)$).

From the cost formulas derived above, it follows that when the number of references stored for a resource is equal to $N-1$, then the cost of locating it reduces to 1 (clearly the absolute minimum when it is not found locally). However, distributing the references to n nodes (e.g., when the

resource changes location) in the network will incur its own cost. In a complete network, in order to distribute n references to a given resource, n links have to be traversed. Thus the total cost per location operation, would have to account for the cost of distributing the references as well. A straightforward way to do this is to divide the cost of distributing n references among all the location operations that take place between two consecutive update operations (updates are done when the resource migrates to a new node). Considering the rates of update and location requests, μ and λ, respectively, the number of location operations between two consecutive update operations would be given by $\frac{\lambda}{\mu}$. Thus we would have to add $\frac{\mu}{\lambda}(n-1)$ to the cost of each location operation, obtaining the following formula for the total cost per location operation,

$$T = \frac{N}{n+1} + \frac{\mu}{\lambda}(n-1) \tag{1}$$

Theorem 2 *If $\frac{4}{N} \leq \frac{\lambda}{\mu} \leq N$, then the number of references that minimizes T is $n_{\min} = \sqrt{\frac{\lambda}{\mu}N} - 1$, and the minimum total cost would be $T_{\min} = 2(\sqrt{\frac{\mu}{\lambda}N} - \frac{\mu}{\lambda})$.*

Proof: By taking the derivative of the cost formula and equating it to zero.

In the above theorem we have considered only cases in which $\frac{4}{N} \leq \frac{\lambda}{\mu} \leq N$, this is due to the fact that for all other cases, the minimum of the cost formula is attained for values of n less than 1 or greater than $N-1$. In those cases, clearly, the optimum strategy would be to keep just one reference or broadcast a reference to the resource to all nodes in the network respectively.

3.2 Extensions to General Networks

The above results are only exact for complete networks. For arbitrary networks, however, we can only provide a lower bound, that is, for a general $G(V,E)$, $T_{\min}(G) \geq T_{\min}(K_{|V|})$. For non-complete networks we can establish the existence of an algorithm R for which $J(R,Q,v) \leq N - n$. Furthermore, if $G(V,E)$ is a hamiltonian graph, and Q is uniform, there is a deterministic algorithm whose cost is

$$C = \frac{N}{n+1}$$

(See [10] for proofs and more discussion).

4 Optimal Resource Finding for Informed Searchers

In this section we investigate the problem of finding the optimal way to conduct a serial search in a general network when the searcher knows the distribution Q. A location algorithm R will be optimal when $J(R,Q,v)$ is minimal for each v. It can be readily seen that for each $v \in V$, there is a certain traversal (not necessarily unique), $s(v) \in C(G,v)$, such that the algorithm R defined by $R_v(s(v)) = 1$ is an optimal one. This will happen when $s(v)$ is such that $C^Q_{s(v)}$ is minimal for all $s \in C(G,v)$. In other words, there is an optimal location finding algorithm which is deterministic. Thus the problem of finding the optimal algorithm reduces to the problem of finding, for each v, an optimal serial traversal in $C(G,v)$.

We will restrict the model to those networks in which $n = 1$. In that case $Q(\{v\})$ becomes the probability that the resource resides at node v, and we will use the notation $\Pi(v)$ instead. For this particular case it can be seen that the cost formula presented in definition 6 gets simplified to the following expression.

$$C^{\Pi}_{s} = \sum_{v \in V} \ell(s_v)\Pi(v) \tag{2}$$

We now present several results concerning the time complexity of choosing the optimal traversal for this particular case. We show that the optimization problem (as it will be presented later) is *NP-hard* for general graphs. This result was also shown by Trummel and Weisinger [11] in a different formulation. We will show the additional result that holds for complete graphs with non-unit edge costs. Notice that the NP-hardness results obtained for our special case are also applicable to the general problem in which the number of references is not fixed.

In the above discussion, the optimal traversal is restricted to be complete (that is, every node in the network has to appear in the traversal). At first sight this may appear as a very restrictive assumption, because it is possible that some nodes have zero probability (thus they do not need to be searched). However this is not the case. For instance, assume that s is an optimal non-complete serial traversal which goes through every node with non-zero probability and leaves out some of the zero-probability nodes. Such a traversal can be completed with a walk that covers the rest of the nodes and which does not add any extra cost according to definition 6. Thus each optimal traversal has at least one corresponding optimal complete traversal, and no generality is lost by considering only the complete ones.

4.1 Complete and General Networks

We will show that the problem of finding an optimal complete serial traversal is *NP-hard* even for complete networks, and it is necessary to simplify the problem to obtain a polynomial algorithm. We will first show, in the next theorem, a polynomial algorithm to solve the optimal traversal problem when the cost function is a constant.

Theorem 3 *Let $G(V,E,c) \equiv K_N$, such that $c(e) = M$ for all $e \in E$. Then, given Π, a traversal $s = (v_0, v_1, \ldots, v_{N-1})$, is optimal if and only if $\Pi(v_i) \geq \Pi(v_{i+1})$*

A direct result of theorem 3 is the existence of an efficient algorithm to find the optimal traversal in a complete graph: it would only have to sort the nodes in decreasing order for the value of the distribution Π (thus, the time complexity would be $O(N \log N)$). If we do not make c constant, however, the problem becomes *NP-hard*, even for the simpler case in which the distribution is uniform.

Theorem 4 *The problem of finding the optimal serial traversal in a distributed system, $G(V, E, c)$, where $G(V, E) = K_N$ and $c(e) = 1$ or 2 for all $e \in E$, and where Π is the uniform distribution, is NP-hard.*

Proof: By reducing the hamiltonian path problem to our problem. See [10].

Thus we see that a small increase in the complexity of the problem renders it computationally intractable. In the next section we will show that for general graphs, the problem is intractable even if the edge cost function is constant.

Theorem 5 *The problem of finding an optimal serial traversal starting at a particular node of a weighted graph, with weight function constant and equal to 1, is NP-hard (see [11]).*

Proof: By reducing the hamiltonian path problem to our problem. See [10].

4.2 Tree Networks

Although, an algorithm to solve the optimal serial traversal problem is computationally intractable for arbitrary graphs, it may be possible to identify a subclass of graphs for which there exists a more efficient algorithm. A heuristic can then be used for general graphs by applying the algorithm to a subgraph in the class which is efficiently solvable. We will study the class of tree structures. An algorithm is presented which is polynomial for a certain class of trees with a restricted number of what we call *frontiers* (to be defined later). Examples of such subclasses of trees are line graphs and star graphs. Furthermore, we will show how to use the algorithm to solve the problem in a bidirectional ring network.

Definition 9 *Let $G(V, E, c)$ be a graph, and let $w_1, w_2 \in \mathcal{W}(G)$, be two walks, $w_1 = (v_0, \ldots, v_m)$ and $w_2 = (u_0, \ldots, u_p)$. If $v_m = u_0$ we will say that w_2 is composable with w_1 and we will define their composition, $w_1 \cdot w_2 = (v_0, \ldots, v_m = u_0, \ldots, u_p)$, which is the sequence resulting from appending w_2 to w_1, without repeating u_0.*

It is straightforward to see that the composition of two walks is also a walk, that is $w_1 \cdot w_2 \in \mathcal{W}(G)$. However the composition of two serial traversals may not be a serial traversal. If s_1 and s_2 are two serial traversals, we will say s_2 is compatible with s_1 if s_2 is composable to s_1 and $s_1 \cdot s_2$ is also a serial traversal.

In a tree, there is a unique simple path between any pair of nodes. We will denote by $L_{u,v}$ the unique simple path joining nodes u and v.

Lemma 3 *Let $G(V, E, c)$ be a tree. Then, given $s = (v_0, \ldots, v_l) \in \mathcal{S}(G)$, $L_{u,v}$ is compatible with s iff $u = v_l$ and $v \notin V(s)$.*

For the rest of this section we will consider rooted trees in which the search starts at the root, and proceeds by expanding the set of searched nodes, which will form a connected neighborhood of the root. The root node of the tree will be represented by the letter r. The next definition formalizes this "neighborhood" concept.

Definition 10 *A frontier of a rooted tree, $G(V, E, c)$, with root r, is an ordered pair $f = (B, v)$ where B contains the leaves of a subtree of G denoted as $ST(B)$.*

The set of nodes visited by a serial traversal s starting at the root will contain the root. Note that the subgraph defined by $V(s)$ is connected and forms a subtree. Thus we can associate a frontier with a given traversal.

Definition 11 *We define the following,*

- *Given a serial traversal starting at the root, $s = (v_0, \ldots, v_l)$, we define the frontier associated with the traversal by $F(s) = (B, v_l)$, where B is the set of leaves of the subtree defined by $V(s)$.*
- *Given a frontier, f, we define $S(f)$ as the set of serial traversals whose frontier is f.*
- *$O(f, \Pi) = \{s \in S(f) | C^{\Pi}_{s'} \geq C^{\Pi}_{s} \text{ for all } s' \in S(f)\}$.*
- *The optimal cost, Ω_f, as , $\Omega_f = C^{\Pi}_{s}$ for $s \in O(f, \Pi)$.*

Lemma 4 *Let $s \in S(G, r)$ such that $F(s) = (B, v)$ and $B \neq \{r\}$. There is a traversal s' such that $F(s') = (B', v')$, $V(s') = V(s) - \{v\}$, and $s = s' \cdot L_{v',v}$.*

If $s \in S(f)$ for some $f = (B, v)$, $B \neq \{r\}$, then, by lemma 4, $s = s' \cdot L_{v',v}$ where $F(s') = f' = (B', v')$ with $ST(B') = V(s') = V(s) - \{v\} = ST(B) - \{v\}$. And we can write

$$C^{\Pi}_{s} = C^{\Pi}_{s'} + \ell(L_{v',v})(1 - \sum_{u \in ST(B')} \Pi(u)) \qquad (3)$$

We can see that the second term in the expression for the cost of s does not depend on the particular s' selected, but only on the frontier f'. This allows us to establish the following result.

Theorem 6 *Let $f = (B, v)$, and let $s \in O(f, \Pi)$ then there is an $f' = (B', v')$ and $s' \in O(f', \Pi)$, such that $s = s' \cdot L_{v',v}$ and $V(s') = V(s) - \{v\}$.*

Based on the previous theorem we can now give an algorithm to find an optimal traversal of a tree. The algorithm finds an optimal serial traversal for each frontier of the tree. The algorithm finds such optimal traversals starting with frontiers $(B, .)$ such that $|ST(B)| = 1$. until it gets to those with $|ST(B)| = N$. It then selects the frontier with minimal cost among those with cardinality N. To find an optimal traversal of cardinality i it uses lemma 4 to find the set of frontiers which are candidates to produce the subtraversal of the optimal traversal in the current frontier.

In the following algorithm we will assume that a certain Π is given. F_i stands for the set of frontiers $(B, .)$ such that $|ST(B)| = i$. S_f will represent an optimal, traversal for frontier f, i.e., $S_f \in O(f, \Pi)$.

Algorithm 1

Precompute $\ell(L_{v,v'})$ for all pairs $\{v, v'\}$;
$f := (\{r\}, r)$; (* note $F_1 = \{f\}$ *)
$\Omega_f := 0$;
$S_f := (r)$;
for $i := 2$ to N **do**
  Find the set F_i, based on F_{i-1};
  for each $f \in F_i$ **do**
  $(B, v) := f$;
  Let $D(B, v)$ be the subset of F_{i-1} whose frontiers $f' = (B', v')$ are such that $ST(B') = ST(B) - \{v\}$;
  $\Omega_f := \min_{f' \in D(B,v)} \{\Omega_{f'} + \ell(L_{v',v})(1 - \Pi(ST(B')))\}$;
$(B_m, v_m) := f_m$; (*frontier yielding the above min*)
  $S_f := S_{f_m} \cdot L_{v_m,v}$;
  endfor
endfor
return(S_f), where f is a frontier in F_N with minimum Ω_f.

For particular subclasses of trees, the algorithm can be further specialized to increase its efficiency. For instance, when the only nodes with non-zero probability are the leaves of the tree, the only frontiers that have to be considered are those which have the same leaf nodes as the original tree.

For a star network with b branches and maximum branch length of h, the maximum number of frontiers will be bounded by $h^b \cdot b$. Thus for constant b the algorithm will run in polynomial time. A path graph is a special case of a star with at most two branches. The subtrees of a path graph will have at most two leaf nodes, each at a different branch from the root. Thus the total number of frontiers will be bounded by N^2, making algorithm 1 polynomial.

4.2.1 Special Cases

There are some special cases for which the structure of the problem makes it possible to obtain more efficient algorithms to find optimal traversals.

Uniform Distributions. In the special case when the distribution is uniform and all the edge costs are equal, any depth-first traversal is optimal.

Ring Networks. For a bidirectional ring we can use a modified version of algorithm 1 to find the optimal traversal. First we observe that in such a network, a complete traversal will leave one edge untraversed. Thus, if s were an optimal traversal and e is one of the unused edges, s would also be an optimal traversal for the line graph obtained by eliminating e from E. Thus the previous algorithm can be modified in the following way:

Algorithm 2

for each $e \in E$ consider $G_e(V, E - \{e\}, c)$
Find an optimal traversal s_e for G_e applying algorithm 1;
endfor
Let $s = s_e$ such that $C_s^{\Pi} = \min_{e' \in E} C_{s_{e'}}^{\Pi}$;
return s

4.2.2 Numerical Examples

Let us consider the tree in figure 1 and assume that the search starts at node 1. We will use the notation $\underline{\Pi}$ to represent the vector $\underline{\Pi} = (\Pi(2), \ldots, \Pi(8))$. When Π is uniform we get the optimal traversal presented in figure 1-(a). We notice that the optimal traversal is a depth-first traversal of the tree. When $\underline{\Pi} = (0.01, 0.01, 0.01, 0.2425, 0.2425, 0.2425, 0.2425)$, that is, the leaf nodes have a higher probability than the internal nodes, the optimal traversal presented in figure 1-(b) goes first to the subtree containing the largest number of leaf nodes, then proceeding with the other subtree. For the same probability distribution, when we make the costs of the edges $\{4, 7\}$ and $\{4, 8\}$ equal to 2, we get the traversal in figure 1-(c), which decides to leave the traversal of the heavier weights until the end. Finally, for the probability distribution $\underline{\Pi} = (0.5, 0.4, 0.002, 0.08, 0.014, 0.002, 0.002)$, we get the optimal traversal in figure 1-(d). This traversal visits higher probability nodes first.

5 Concluding Remarks

We have seen that in the best case, the average cost incurred by a searcher node which has no knowledge about the location of the resource, is at best of the order of $\sqrt{\frac{\mu}{\lambda} N}$. A cost of $\sqrt{\frac{\mu}{\lambda} N}$ may not be cheap in large networks. The conclusion that we draw is that it is necessary to possess some more knowledge about the location of a resource to make the process of finding it efficient.

We have also studied the problem of finding an optimal serial traversal when the searcher has the distribution for the well-informed set at request time. In particular we have looked at the case when the well-informed set is a singleton. The results show that such a problem is NP-hard even for complete graphs in which all links do not have the same

cost (in fact, when there are only two different values for the link costs). Thus the existence of an efficient algorithm to find an optimal serial traversal is very unlikely. In the last sections we have shown that the problem can be solved in polynomial time for some classes of trees, and based on it we have shown a polynomial algorithm for ring networks.

References

[1] A. D. Birrel, R. Levin, R. M. Needham, and M. D. Schroeder, "Grapevine: an exercise in distributed computing," *Communications of the ACM*, vol. 25, pp. 260–274, April 1982.

[2] D. Terry, "Caching hints in distributed systems," *IEEE Transactions on Software Engineering*, vol. SE-13, pp. 48–54, January 1987.

[3] S. Mullender and P. Vitányi, "Distributed match-making for processes in computer networks," in *Fourth ACM Symposium on the Principles of Distributed Computing*, (Minacki, Ontario), ACM, August 1985.

[4] R. Fowler, "The complexity of using forwarding addresses for decentralized object finding," in *Fifth ACM Symposium on the Principles of Distributed Computing*, (Calgary, Alberta, Canada), pp. 108–120, ACM, August 11–13 1986.

[5] R. Fowler, "Decentralized object finding using forwarding addresses," PhD Thesis 85-12-1, University of Washington, December 1985.

[6] N. Megiddo, S. Hakimi, M. Garey, D. Johnson, and C. Papadimitriu, "The complexity of searching a graph," *J. ACM*, vol. 35, pp. 18–44, Jan. 1988.

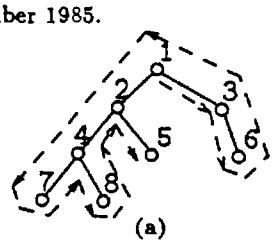
[7] T. D. Parsons, "Pursuit–evasion in a graph," in *Theory and Applications of Graphs*, (Y. Alavi and D. Lick, eds.), pp. 426–441, Berlin: Springer–Verlag, 1976.

[8] M. Ahamad, M. Ammar, J. Bernabéu–Aubán, and Y. Khalidi, "Using Multicast Communication to Locate Resources in a LAN-Based Distributed System," in *Proceedings of the 13th Conference on Local Computer Networks*, IEEE, October 1988.

[9] J. Bernabeu, M. Ammar, and M. Ahamad, "Optimal selection of multicast groups for resource location in a distributed system," in *Proceedings of IEEE INFOCOM*, IEEE, 1989.

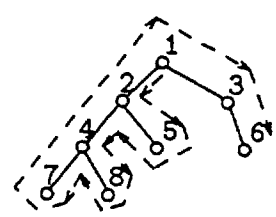
[10] J. Bernabeu-Auban, M. Ahamad, M. Ammar, "Resource finding in store-and-forward networks," School of Information and Computer Science, Georgia Institute of Technology, Report GIT-ICS-89/04, February 1989.

[11] K. Trummel and J. Weisinger, "The complexity of the optimal searcher path problem," *Operations Research*, vol. 34, pp. 324–327, March–April 1986.



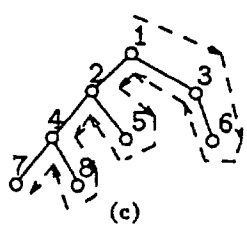
(a)

Uniform Π
$c(e) = 1$ for all $e \in E$.



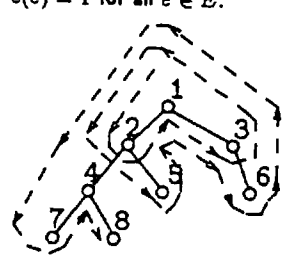
(b)

$\Pi(v) = 0.01$, for $v \in \{2,3,4\}$
$\Pi(v) = 0.2425$ for $v \in \{5,6,7,8\}$
$c(e) = 1$ for all $e \in E$.



(c)

$\Pi(v) = 0.01$, for $v \in \{2,3,4\}$
$\Pi(v) = 0.2425$ for $v \in \{5,6,7,8\}$
$c(\{4,7\}) = c(\{4,8\}) = 2$
$c(e) = 1$ for all other $e \in E$



(d)

$\underline{\Pi} = (0.5, 0.4, 0.002, 0.08, 0.014, 0.002, 0.002)$
$c(e) = 1$ for all $e \in E$

Figure 1: Example of optimal traversals in tree networks

Multi-Dimensional Voting: A General Method for Implementing Synchronization in Distributed Systems*

Shun Yan Cheung *Mustaque Ahamad* *Mostafa H. Ammar*

School of Information and Computer Science

Georgia Institute of Technology

Atlanta, Georgia 30332

Abstract

We introduce a new concept, *multi-dimensional voting*, in which the vote and quorum assignments are k-dimensional vectors of non-negative integers and each dimension is independent of the others. Multi-dimensional voting is more powerful than traditional weighted voting because it is equivalent to the general method for achieving synchronization in distributed systems which is based on coteries (set of groups of nodes) but its implementation is easier than coteries. We describe an efficient algorithm for finding a multi-dimensional vote assignment for any given coterie and show examples of its use. We also show how multi-dimensional voting can be used to easily implement novel algorithms for synchronizing access to replicated data or to ensure mutual exclusion. These algorithm cannot be implemented by traditional weighted voting.

1 Introduction

Distributed systems offer many advantages, including resource sharing and fault-tolerance. The latter can be achieved by replicating a resource at nodes with independent failure modes. Replication can also improve performance when load is shared among the nodes that have instances of a resource. In many applications, users need to synchronize access to shared resources. For example, when data is replicated to improve its availability, updating the file requires mutually exclusive access. This is necessary for maintaining the consistency of the data. The synchronization technique should work in the presence of node and communication failures.

An operation that requires mutual exclusion can be executed if permission can be obtained from a group of nodes. In general, a node can execute the operation if permission can be obtained from any one group in a set of intersecting groups [1]. Such a set is called a *coterie* in [2]. For reading and writing of replicated data when several readers are allowed to access the data concurrently, read and write coteries can be defined in a similar way [3]. Another well-known synchronization method is weighted voting [4] which is a generalization of the majority consensus method [5]. In voting, each node is assigned a number of votes and each operation must obtain a pre-defined quorum of votes before it is allowed to execute to completion. Voting can be used for achieving mutual exclusion and synchronizing reading and writing of replicated data. In mutual exclusion, each operation must obtain a majority of the votes assigned before it can proceed. In reading and writing, the read and write quorums must be such that their sum is more than the total number of votes and the write quorum is at least a majority of all votes.

Voting is appealing because it is flexible and can be easily implemented. Each node in voting stores its assigned vote and when it wants to execute an operation that requires q votes, it communicates with other nodes to request their votes. The execution of the operation can proceed if the sum of the votes received is at least q. In contrast, in a system that uses coteries, operations must know all the groups of the coterie and test if the nodes that responded positively to its request form a group of the coterie. Voting is also more flexible. Adding or removing a node requires only a change of the quorum and assigning the proper number of votes to the new node. In a coterie-based system, adding and removing a node may cause the addition and deletion of numerous groups. However, Garcia-Molina and Barbara proved in [2] that the method of coteries is more general than voting by showing coteries which cannot be obtained from any vote assignment. Coteries that are not obtained from vote assignments can be used to achieve better performance by reducing the number of messages. For example, structured coteries as those used in the methods described in [6, 7] have lower communication cost and they cannot be implemented by voting.

We present in this work a new voting based method that is as powerful as the method of coteries and has the flexibility and ease of implementation of voting. In multi-dimensional voting, the vote assignment to each node and the quorums are k-dimensional vectors of non-negative integers. Each dimension of the vote and quorum assignment is similar to voting and the quorum requirements in different dimensions can be combined in a number of ways.

*This work was supported in part by NSF grants NCR-8604850 and CCR-8806358.

This makes multi-dimensional voting more powerful than standard voting. We will discuss a number of applications which can be implemented with multi-dimensional voting but not with standard voting.

We show that every coterie can be represented by a multi-dimensional vote assignment and present an efficient algorithm for finding one. The use of the algorithm is demonstrated in finding multi-dimensional vote assignments for coteries that cannot be obtained from standard vote assignments. We also discuss how reading and writing of replicated data can be synchronized using multi-dimensional voting.

The paper is organized as follows. In Section 2, we will introduce the concept of a multi-dimensional vote assignment and in Section 3, we present an algorithm for finding multi-dimensional vote and quorum assignments for sets of groups with a certain property which is satisfied by coteries. Sections 4 and 5 discuss the use of multi-dimensional voting for mutual exclusion and reading and writing of replicated data respectively. We conclude the paper in Section 6.

1.1 Related Work

Coteries and voting have been used to synchronize access to replicated data and the methods are called *replica control protocols*. Related work on these protocols include many dynamic replica control methods that are derived from voting [8, 9, 10, 11, 12] and these methods achieve very high data availability. Available copies and regeneration methods, such as the ones described in [13] and [14], achieve even higher data availability using the same number of copies but data consistency cannot be guaranteed if the network can partition. To reduce storage space for copies, the methods described in [15] and [16] can be used, but data availability will also be compromised. The mutual exclusion method presented in [6] uses coteries that are derived from a binary tree structure. A comprehensive survey of replica control methods is presented in [17].

The problem of enumerating coteries so that performance can be optimized by choosing the best one has been addressed by several researchers. In [2], an algorithm is described that can be used to generate a subset of the mutual exclusion coteries which includes all coteries obtained from vote assignments. The authors presented an algorithm in [3] to generate all vote and quorum assignments that need to be considered in optimizing reading and writing of replicated data. The method presented in [18] generates a subset of the mutual exclusion coteries obtained from vote assignments.

Optimization using availability as the performance measure has been considered in a number of works. Barbara and Garcia-Molina showed in [19] that the vote assignment which allocates one vote to each node will maximize availability for mutual exclusion if the nodes are uniform. In a related work [20], Ahamad and Ammar studied availability and response time of read and write operations for systems of uniform nodes and in [21], the authors presented a scheme that can improve response time through a higher degree of load sharing. The performance of the available copies replica control protocol and its variants is studied in [22] and [23].

2 Multi-Dimensional Voting

We consider a distributed system of N nodes which are numbered as 1, 2, ..., N. In multi-dimensional (MD) voting, the vote value assigned to a node and the quorum are k-dimensional vectors of non-negative integers. Formally, the MD vote assignment $V_{N,k}$ is a Nxk matrix where $v_{i,j}$ represents the vote assignment to node i in the j^{th} dimension and $v_{i,j} \geq 0$ for $i = 1, 2, \ldots, N$ and $j = 1, 2, \ldots, k$. The votes assigned in the various dimensions are independent of each others. The quorum assignment $\underline{q}_k = (q_1, q_2, \ldots, q_k)$ is a k-dimensional integer vector, where $q_j > 0$, for $j = 1, 2, \ldots, k$. In addition, a number ℓ, $1 \leq \ell \leq k$, is defined which is the number of dimensions of vote assignments for which the quorum must be satisfied. Thus, there are *two* levels of requirements: vote and dimension level. At the vote level, the number of votes must be greater than or equal to the quorum requirement in the same dimension and at the dimension level, the number of dimensions for which a quorum is collected must be greater than or equal to ℓ. As we show in the next section, this extra level of flexibility makes MD-voting more powerful than standard voting. We denote MD-voting with quorum requirement in ℓ of k dimensions as MD(ℓ, k)-voting and the term SD-voting (single dimensional voting) will refer to the standard voting method described in [4]. In fact, MD(1,1)-voting is the same as SD-voting.

Synchronization methods developed from MD-voting operate in a similar manner as SD-voting. Each node stores its vote which consists of k integers and each operation has a quorum requirement for each dimension and the value of ℓ. An operation requests permission from the nodes by sending a voting request to them. When a node receives a vote request, it votes reject if it wants to disallow the operation to proceed (e.g., due to locking conflict) or replies with its vote in all dimensions. Each operation maintains k independent variables which accumulate the votes received in each dimension. When a response containing a vote is received, the operation adds the vote in each dimension to the appropriate variable and when the sums in at least ℓ variables are greater than or equal to the quorum in the corresponding dimensions, the operation can proceed.

Each node must store k integers and the voting messages used will also contain all the integers. When a large number of dimensions is used, the voting messages can be long and in the next section, we will present a method for finding MD vote and quorum assignments that use a relatively small number of dimensions.

3 Finding a Multi-Dimensional Vote Assignment

3.1 Definitions and Notation

Let $U = \{1, 2, \ldots, N\}$ be the universe set of all nodes and we will refer to sets of nodes as *groups*. A set of groups Q has the *minimality* property [2] if,

$$\forall G, H \in Q: \; G \not\subseteq H$$

The synchronization requirements define what groups are included in the set. For example, if mutual exclusion is desired, this set is called a coterie and any two of its members must have a non-empty intersection (see Section 4).

A number of the sets that have the minimality property can be represented by SD-voting. Each node i in SD-voting is assigned v_i votes ($1 \leq i \leq N$) where v_i is a non-negative integer and a quorum q is defined, such that nodes in each group of the set have at least q votes. Specifically, with the vote assignment $\underline{v} = (v_1, v_2, \ldots, v_N)$, the members of the set defined by $(\underline{v}, q)$ are tight groups of nodes which have at least q votes. A group G is *tight* with respect to quorum q if,

- $\sum_{g \in G} v_g \geq q$ and,
- any proper subset of G has less than q votes

The set of tight groups Q defined by $(\underline{v}, q)$ is,

$$Q = \{G \mid G \text{ is a tight group with respect to quorum } q\}$$

and this set has the minimality property since if there would exist $G, H \in Q$ such that $G \subset H$, then H would not be tight. The vote and quorum assignment $(\underline{v}, q)$ defines a *unique* set of tight groups which has the minimality property. For instance, the vote assignment (1,1,1) to a three node system and the quorum requirement of 2 votes defines the set of tight groups {{1,2},{1,3},{2,3}}. The same set can be represented by the vote assignment (2,2,3) and $q = 4$ and hence a set may not have a unique vote and quorum assignment.

An MD(ℓ, k) vote and quorum assignment also defines a unique set of tight groups in a similar manner as SD-voting. A group G is a *tight* group in MD(ℓ, k)-voting with respect to quorum requirement $\underline{q}$ if

- $\sum_{g \in G} v_{g,j_i} \geq q_{j_i}$ for ℓ distinct dimensions $j_1, j_2, \ldots, j_\ell$ and,
- any proper subset of G satisfies quorum requirement in strictly less than ℓ dimensions

The set $Q_{\ell,k}(V_{N,k}, \underline{q}_k)$ of tight groups represented by the MD(ℓ, k) vote and quorum assignment $(V_{N,k}, \underline{q}_k)$ is,

$$Q_{\ell,k}(V_{N,k}, \underline{q}_k) = \{G \mid G \text{ is a tight group in MD}(\ell, k)\text{-voting with respect to } \underline{q}\}$$

Similar to SD-voting, the same set of tight groups can be represented by different MD(ℓ, k) vote and quorum assignments and the groups are also minimal. The set of tight groups for the special cases where $\ell = 1$ (any dimension) and $\ell = k$ (all dimensions) can be given as follows,

$$Q_{1,k}(V_{N,k}, \underline{q}_k) = \{G \mid G \text{ is a tight group such that: } \exists j;\ 1 \leq j \leq k: \sum_{g \in G} v_{g,j} \geq q_j\}$$

$$Q_{k,k}(V_{N,k}, \underline{q}_k) = \{G \mid G \text{ is a tight group such that: } \forall j;\ 1 \leq j \leq k: \sum_{g \in G} v_{g,j} \geq q_j\}$$

In MD(1,k)-voting, an operation can proceed if quorum is available in any dimension and in MD(k, k)-voting, quorum requirements in all dimensions must be satisfied. For MD(1,k)-voting, we can also write,

$$Q_{1,k}(V_{N,k}, \underline{q}_k) = \{G \mid G \in \bigcup_{j=1}^{k} C_j \wedge \forall H \in \bigcup_{j=1}^{k} C_j: \; H \not\subset G\}$$

where C_j is the set of tight groups defined by the j^{th} dimension of vote and quorum assignment, i.e, $Q_{1,k}(V_{N,k}, \underline{q}_k)$ is all the minimal groups in $\bigcup_{j=1}^{k} C_j$.

Table 1 presents a two-dimensional vote and quorum assignment to a system of four nodes. The sets C_1 and C_2 are the sets of tight groups corresponding to the first and second dimension of the MD vote and quorum assignment, respectively.

$$V_{4,2} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 0 & 2 \end{pmatrix}, \quad \underline{q}_2 = (2, 3)$$

$C_1 = \{\{1,2\},\{1,3\},\{2,3\}\}$

$C_2 = \{\{1,4\},\{2,4\},\{3,4\},\{1,2,3\}\}$

$Q_{1,2}(V_{4,2}, \underline{q}_2) = \{\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,4\},\{3,4\}\}$

$Q_{2,2}(V_{4,2}, \underline{q}_2) = \{\{1,2,3\},\{1,2,4\},\{1,3,4\},\{2,3,4\}\}$

Table 1: An example of multi-dimensional vote assignment

3.2 The Existence of MD Vote Assignments

We show that any set Q that has the minimality property can be represented by an MD(1,k) vote and quorum assignment.

Lemma 3.1: Let Q be a set of groups such that

$$\forall\, G, H \in Q: \quad G \not\subseteq H$$

Then Q can be represented be an MD(1,k) vote and quorum assignment where $k = |\,Q\,|$.

Proof: Let $Q = \{G_1, G_2, \ldots, G_k\}$ so that $|\,Q\,| = k$. We construct the following k-dimensional vote assignment: the vote value of node i in the j^{th} dimension, $i = 1, 2, \ldots, N$ and $j = 1, 2, \ldots, k$ is given by,

$$v_{i,j} = 1 \qquad \text{for } i \in G_j$$
$$v_{i,j} = 0 \qquad \text{for } i \notin G_j$$

with $q_j = |\,G_j\,|$. We will show that $Q = Q_{1,k}(V_{N,k}, \underline{q}_k)$.

From the construction of the MD(1,k) vote and quorum assignment that yields $Q_{1,k}(V_{N,k}, \underline{q}_k)$, it is trivially true that $H \in Q \implies H \in Q_{1,k}(V_{N,k}, \underline{q}_k)$, i.e., $Q \subseteq Q_{1,k}(V_{N,k}, \underline{q}_k)$. (When $H = G_j$, votes from nodes in H satisfy the quorum requirement in the j^{th} dimension and H is also tight.) Consider the j^{th} dimension of the MD vote assignment that is derived from the group $G_j \in Q$. The set of groups C_j represented by this dimension is equal to $\{G_j\}$ and since $Q_{1,k}(V_{N,k}, \underline{q}_k)$ is all the minimal groups in $\bigcup_{j=1}^{|Q|} C_j$ (see (1)), we have $Q_{1,k}(V_{N,k}, \underline{q}_k) \subseteq Q$.□

Lemma 3.1 guarantees that an MD(1,k) vote and quorum assignment can be found for any set of minimal groups Q. In the constructive proof, since each group is represented by a separate dimension, the number of dimensions used is equal to $|\,Q\,|$, which may be large. We present in what follows a technique that uses the fact that several groups may be representable by a single dimension of an MD vote and quorum assignment. Therefore, in practice, the number of dimensions needed to obtain an MD(1,k) vote assignment could be less than $|\,Q\,|$.

3.3 Finding an SD vote assignment

In [3], a technique is described for testing if a set of groups Q is SD vote assignable. The following linear program, LP(Q), is setup using the groups in Q,

$$\text{Min:} \sum_{i=1}^{N} v_i + q$$

$$\text{s.t.:} \forall\, G \in Q: \quad \sum_{g \in G} v_g \geq q \qquad (1)$$

$$\forall\, H \in \overline{sup}(Q\,|\,U) \cup psub(Q): \quad \sum_{h \in H} v_h \leq q - 1 \qquad (2)$$

$$v_i \geq 0, \quad i = 1, 2, \ldots, N$$

$$q \geq 1$$

where,

- $\overline{sup}(Q\,|\,U)$ is the set of all groups that are subsets of U and not supersets of any group in Q, and
- $psub(Q)$ is the set of all groups that are proper subsets of the groups in Q.

If LP(Q) does not have a feasible solution then Q is not SD vote assignable, otherwise a feasible solution (which is rational) can be converted to an integer vote and quorum assignment.

Unlike [3], we are dealing here with sets Q with the minimality property. This allows us a further refinement of LP(Q) which is a result of the following lemma.

Lemma 3.2: Let Q be a set of groups satisfying the minimality property, i.e., $\forall\ G, H \in Q: \quad G \not\subseteq H$. Then, $psub(Q) \subseteq \overline{sup}(Q\,|\,U)$.

Proof: The proof is by contradiction.

Assume there is a group X that is in $psub(Q)$ but not in $\overline{sup}(Q\,|\,U)$. Since $X \in psub(Q)$, X is a proper subset of some group $A \in Q$. Furthermore, $X \notin \overline{sup}(Q\,|\,U)$ so it is a superset of some (other) group B such that $B \in Q$. However, then Q would violate the minimality property because the above facts imply that $B \subset A$. This contradicts the premise.□

We can thus substitute constraint (2) in LP(Q) with,

$$\forall\, H \in \overline{sup}(Q\,|\,U): \quad \sum_{h \in H} v_h \leq q - 1 \qquad (3)$$

3.4 Algorithm for Finding MD(1,k) Vote and Quorum Assignment

Here we extend the procedure described in subsection 3.3 to find an MD(1,k) vote and quorum assignment for a set of groups Q satisfying the minimality property. Our algorithm (illustrated in Figure 1) finds an MD(1,k) vote assignment by testing to see if Q is SD vote assignable. If not, groups are systematically removed from Q until the groups that remain form an SD vote assignable set. The votes and quorum obtained from the solution form the assignment in the first dimension. The set of groups removed from Q to make it SD vote assignable are then used as input to a second iteration to find the second dimension of vote and quorum assignment. This is repeated until no more groups remain. Since a set with one group is always vote assignable, in each iteration at least one group of Q is represented by the solution found and the algorithm is guaranteed to terminate. An efficient implementation of the algorithm is presented in [24]. Sections 4 and 5 contain examples of the application of our algorithm.

4 MD-Voting for Mutual Exclusion

The problem of mutual exclusion arises in many applications where a process must acquire exclusive access to

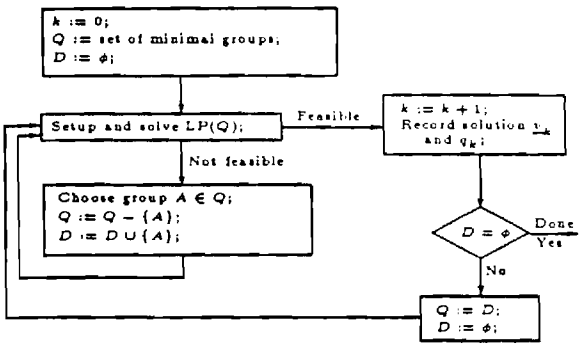


Figure 1: Algorithm for finding MD(1,k) vote and quorum assignment

a shared resource. In distributed systems, the synchronization method used must tolerate node and link failures. The general method for achieving synchronization in a distributed system is the use of *coteries*. The definition of a coterie is given in [2] and it is repeated here for completeness.

Definition 4.1: *Coterie* [2]. A set of groups Q is a *coterie* under U iff

1. $G \in Q \implies G \subseteq U \wedge G \neq \phi$
2. (Intersection property) $\forall\, G, H \in Q:\ G \cap H \neq \phi$
3. (Minimality property) $\forall\, G, H \in Q:\ G \not\subseteq H$

A process synchronizes with other processes by obtaining permission from nodes that form a group of the coterie. A node gives permission to only one request at a time and the other requests are kept pending until the request that was given permission completes. The intersection property guarantees that only one process will succeed at a time and mutual exclusion is achieved. However, in the general case, implementation of the method based on coteries could be complex because a coterie can be exponential in size. It will require that processes keep a list of the groups in the coterie and a comparison of the responses against this list is required to determine if a process can proceed.

SD-voting can also be used to achieve mutual exclusion when the quorum used is a majority of the votes. The SD vote assignment to the nodes uniquely determines a coterie and we will call coteries that have an SD-voting equivalent *SD-vote assignable*. There exist coteries that cannot be obtained from SD vote assignments, thus the method of coteries is more general than SD-voting. SD-voting also requires a relatively large number of nodes to participate in the execution of the protocol. For example, to achieve mutual exclusion, nodes that have more than half the votes must participate. Consequently, each mutual exclusion request generates a large number of messages which have a significant impact on response time. Non SD-vote assignable coteries can achieve mutual exclusion using a lower number of messages (e.g. [7], [6]). However, non SD-vote assignable coteries cannot be implemented by using SD-voting. The following corollary shows that MD-voting is as general as coteries and can be used to implement mutual exclusion methods with the desirable properties of SD-voting.

Corollary 4.1: A mutual exclusion coterie Q of N nodes can be represented by an MD(1,k) vote and quorum assignment.

Proof: Since Q satisfies the minimality property, the claim follows from Lemma 3.1.

Table 2 shows a coterie which was described in [2] and it was shown to be non SD-vote assignable. The table presents an MD(1,4) vote and quorum assignment for the coterie. (Notice that the number of dimensions is smaller than the number of groups in the coterie which is 7.)

$Q = \{\{12\},\{134\},\{135\},\{146\},\{156\},\{236\},\{245\}\}$

$$V_{6,4} = \begin{pmatrix} 2 & 0 & 2 & 2 \\ 3 & 1 & 0 & 0 \\ 0 & 1 & 0 & 2 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 2 & 0 \end{pmatrix}, \quad \underline{q}_4 = (5,3,5,5)$$

Table 2: A non SD-vote assignable coterie and its multi-dimensional vote assignment

The structured coterie approach organizes nodes in some logical structure and groups of the coterie are derived from this structure. In [6] the nodes are organized into a binary tree structure and groups of nodes that form a path from the root to a leaf define a group in the coterie. If some node on a path fails, it is replaced by two paths starting from the children of the failed node. Table 3 shows a 6-dimensional vote assignment for the coterie that is derived from a binary tree of depth three. The tree-based method can achieve mutual exclusion using as few as $\log(N)$ messages when there are few failures.

5 MD-Voting for Reading and Writing Replicated Data

We assume data is fully replicated at N different nodes and the copies of data are numbered as 1, 2, ..., N. We will examine the use of MD-voting to implement read and write coteries which correspond to replica control protocols that cannot be implemented using SD-voting. We will assume that version numbers are used to identify the most recently updated replicas.

A read operation returns some value and a write operation installs a new value. Proper synchronization is achieved if a read operation returns the value installed by

Q = {{124},{125},{136},{137},{145},{167},{2346},{2347}, {2356},{2357},{2467},{2567},{3456},{3457},{4567}}

$$V_{7,5} = \begin{pmatrix} 0 & 0 & 0 & 2 & 2 \\ 0 & 2 & 2 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 2 & 0 & 2 & 0 & 1 \\ 2 & 2 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}, \quad \underline{q}_5 = (6,6,6,4,4)$$

Table 3: A tree-based coterie and its multi-dimensional vote assignment

the last write operation and two write operations are not executed concurrently. In general, synchronization of read and write operations can be ensured by requiring that each operation obtain permission of a group of nodes and the groups used by conflicting operations have non-empty intersection. Minimal groups of nodes that can allow a read and write operation to complete are called read and write groups respectively. The read and write coteries R and W are the sets of read and write groups used. (W is a coterie and R is its anti-coterie [25].) The synchronization requirements given above are satisfied if,

1. (Read/write intersection property) $\forall\, G \in R, H \in W$: $G \cap H \neq \phi$, and
2. (Write/write intersection property) $\forall\, G, H \in W : G \cap H \neq \phi$.

R and W have the minimality property and W also has the intersection property. Since read operations can be executed concurrently, the set R need not satisfy the intersection property, i.e., R is in general not a coterie which is used for enforcing mutual exclusion. For a given R, to maximize write availability, W equals the *maximal* set of minimal groups that have read/write and write/write intersection property. The set W, in general, is not unique for a given R. For example, let $R = \{\{1,2\},\{3,4\}\}$, then the sets {{1,3},{1,4},{2,3,4}} and {{1,3},{2,3},{1,2,4}} can both be used as write coteries.

A replica control protocol corresponds to a read and a write coterie which satisfy the synchronization requirements. Thus, in the general case, consistency of replicated data is maintained using two possibly different sets of groups (one for reading and the other for writing) which have the minimality property. It is straitforward to see that an MD vote assignments can be obtained for each of them, one is used for reading and the other one is used for writing. In the general case, the MD vote assignments obtained for the read and write coteries may be different. Consequently, a node must use the appropriate MD vote assignment (based on the type of the request) to vote on each request. Thus, votes obtained for a read request cannot be used for a write request. Although it is feasible to implement read and write coteries with separate MD vote and quorum assignments, it is simpler and more efficient to allow votes obtained for reading to be augmented to a quorum for writing because transactions usually read the data before updating them. This will be similar to SD-voting where the *same* vote assignment is used to define both read and write coteries. In the next subsection, we describe a replica control protocol that uses a single MD vote assignment to define both read and write coteries and allows a read quorum to be augmented when the read data items are also updated.

5.1 A Replica Control Protocol Based on MD-Voting

In the design of a replica control protocol, an appropriate read coterie that provides high read performance is chosen and the corresponding write coterie is computed to satisfy the synchronization requirements. In general, the read coterie can be represented by an MD(ℓ, k) vote and read quorum assignment. Let $V_{N,k}$ and $\underline{r}_k = (r_1, r_2, \ldots, r_k)$ be the vote and read quorum assignment for an MD(ℓ, k)-voting system used in reading and $Q_{\ell,k}(V_{N,k}, \underline{r}_k)$ represents the set of minimal groups defined by the assignment. To allow write operations to synchronize using the same vote and read quorum assignment, we define the write quorum $\underline{w}_k = (w_1, w_2, \ldots, w_k)$ to be,

$$w_j = \sum_{i=1}^{N} v_{i,j} - r_j + 1, \qquad \text{for } j = 1, 2, \ldots, k$$

We *do not* require that $w_j \geq \left\lceil \frac{\sum_{i=1}^{N} v_{i,j}+1}{2} \right\rceil$ votes, for $j = 1, 2, \ldots, k$. The write quorum w_j will only ensure that groups that satisfy the write requirement intersect with all read groups of the j^{th} dimension of the MD vote assignment. Since the read coterie is defined by MD(ℓ, k)-voting, we must use MD($k - \ell + 1, k$)-voting for writing to ensure that the read/write intersection property holds. Let $Q_{k+1-\ell,k}(V_{N,k}, \underline{w}_k)$ be the set of tight groups represented by the MD($k + 1 - \ell, k$) vote and *write* quorum assignment. The following lemma shows that $Q_{\ell,k}(V_{N,k}, \underline{r}_k)$ and $Q_{k+1-\ell,k}(V_{N,k}, \underline{w}_k)$ have the read/write intersection property.

Lemma 5.1:

$$\forall\, G \in Q_{\ell,k}(V_{N,k}, \underline{r}_k),\ H \in Q_{k+1-\ell,k}(V_{N,k}, \underline{w}_k) :\ G \cap H \neq \phi$$

Proof:

Let G and H be two arbitrary groups in $Q_{\ell,k}(V_{N,k}, \underline{r}_k)$ and $Q_{k+1-\ell,k}(V_{N,k}, \underline{w}_k)$ respectively. Since $(\ell) + (k + 1 - \ell) > k$, there is some dimension s such that,

$$\sum_{g \in G} v_{g,s} \geq r_s$$

and

$$\sum_{h \in H} v_{h,s} \geq w_s$$

Since $r_s + w_s > \sum_{i=1}^{N} v_{i,s}$, there must be a common node in G and H and hence $G \cap H \neq \phi$.□

Although the sets $Q_{\ell,k}(V_{N,k}, \underline{r}_k)$ and $Q_{k+1-\ell,k}(V_{N,k}, \underline{w}_k)$ have the intersection property which is necessary for read/write synchronization, $Q_{k+1-\ell,k}(V_{N,k}, \underline{w}_k)$ may not be a write coterie for $Q_{\ell,k}(V_{N,k}, \underline{r}_k)$ because it may not have the write/write intersection property which is required when version numbers are used. To achieve this, we can augment each group of $Q_{k+1-\ell,k}(V_{N,k}, \underline{w}_k)$ to include a group of $Q_{\ell,k}(V_{N,k}, \underline{r}_k)$. We define the write coterie W which is derived from $Q_{\ell,k}(V_{N,k}, \underline{r}_k)$ and $Q_{k+1-\ell,k}(V_{N,k}, \underline{w}_k)$ in the following way:

$$W = \{A \cup B \mid A \cup B \text{ is minimal and } A \in Q_{\ell,k}(V_{N,k}, \underline{r}_k), B \in Q_{k+1-\ell,k}(V_{N,k}, \underline{w}_k)\}$$

It can be easily seen that when $Q_{\ell,k}(V_{N,k}, \underline{r}_k)$ and W are used for reading and writing, both the read/write and write/write intersection properties are satisfied. The latter property follows from the read/write intersection of groups in $Q_{\ell,k}(V_{N,k}, \underline{r}_k)$ and $Q_{k+1-\ell,k}(V_{N,k}, \underline{w}_k)$. The replica control protocol used is as follows: when reading, the operation obtains a read quorum in at least ℓ dimensions and when writing, it must obtain a read quorum *and* a write quorum in ℓ and $k+1-\ell$ dimensions, respectively. If the writing of the data is followed by its read (which is the typical case), the write operation only needs to obtain a write quorum and the method thus allows the read quorum to be augmented.

A special case of the protocol is when MD(1,k)-voting is used for reading. Then, we can use the method in Section 3 to find an MD(1,k) vote and read quorum assignment for the read coterie. The corresponding write coterie will be represented by using an MD(k,k) vote and write quorum assignment. In this case, the read operation can proceed if it can obtain a read quorum in any one dimension. If the transaction wishes to update the data after reading it, the vote received for the read request must be supplemented with additional votes such that in each dimension the number of votes received is greater than or equal to the write quorum for that dimension. The general case where MD(ℓ,k)-voting (arbitrary ℓ) is used, is more difficult as we do not yet have an algorithm to find an MD(ℓ,k) assignment when $\ell \neq 1$. However, if the read coterie used is derived from some logical structure, such as the example described in the next subsection, we may be able to use the structure to formulate an MD assignment.

5.2 Example

The structured coterie concept can also be applied to synchronize reading and writing of replicated data. Similar to mutual exclusion, the resulting read and write coteries may not be SD-vote assignable. Structured read and write coteries can achieve higher load sharing than SD-voting which results in lower response times. Also, the number of messages used in the execution of each operation is reduced.

The replica control method presented in [21] organizes the nodes of the system into a logical grid consisting of m rows and n columns. The read coterie consists of groups of m nodes where one node is selected from each column and a write group consists of nodes in a read group and all nodes in a column of the grid (the coteries used in this method are generally not SD-vote assignable). The simulation study in [21] showed that the response times of transactions in systems using the grid protocol are significantly lower than those that use SD-voting for the same number of nodes. Also, an increase in the number of nodes in a system using SD-voting will not result in much reduction in response time because the load is not shared effectively. Systems using the grid protocol have higher maximum throughput and lower response time.

In [21] we have used coteries to implement the grid protocol. Each operation knows the topology and the position of the nodes in the grid. An operation checks whether the collection of responses constitutes a group that can permit it to proceed. The read and write coteries used in the grid protocol can be represented using MD-voting. In fact, a single MD vote assignment can be used to represent both coteries. The MD vote assignment used for an mxn grid network consists of n dimensions and a node i has $v_{i,j} = 1$ if it is in column j, otherwise $v_{i,j} = 0$, $j = 1, 2, \ldots, n$. The read and write quorums used are $\underline{r}_n = (1, 1, ..., 1)$ and $\underline{w}_n = (m, m, ..., m)$, respectively and MD($n,n$) and MD(1,$n$)-voting is used for reading and writing, respectively. Using the MD-voting implementation of the grid protocol, operations do not need to know the topology of the grid. The read coterie $Q_{n,n}(V_{N,n}, \underline{r}_n)$ consists of groups of nodes with exactly one node from each column and the set $Q_{1,n}(V_{N,n}, \underline{w}_n)$ consists of groups of all nodes in a column of the grid. Groups of the write coterie thus consist of a read group and all nodes in a column of the grid. For instance, a 2x3 grid system in Figure 2 will have the vote assignment given in Table 4. The read and write coteries used are {{1,2,3}, {1,2,6}, {1,5,3}, {1,5,6}, {4,2,3}, {4,2,6}, {4,5,3}, {1,5,6}} and {{1,4,2,3}, {1,4,2,6}, {1,4,5,3}, {1,4,5,6}, {2,5,1,3}, {2,5,1,6}, {2,5,4,3}, {2,5,4,3}, {3,6,1,2}, {3,6,1,5}, {3,6,4,2}, {3,6,4,5}}, respectively.

6 Conclusion

In this paper, we have introduced the concept of a multi-dimensional vote and quorum assignment which is a generalization of standard voting. In multi-dimensional voting, the vote assigned to a node and the quorum assignment are vectors of non-negative integers and each dimension is similar to standard voting. We have shown that any set of minimal groups can be represented by multi-dimensional voting. We have also shown that it is more general than standard voting and is as powerful as the coterie concept, which is the general approach for achieving mutual ex-

clusion in distributed systems. Multi-dimensional voting has the advantage that it is flexible and can be easily implemented. We have developed an efficient algorithm for finding a multi-dimensional vote and quorum assignment for any set of minimal groups and its use was shown by finding multi-dimensional vote assignments for some non SD-vote assignable coteries. We also discuss the use of multi-dimensional voting to synchronize reading and writing of replicated data and showed an example of its use to represent the read and write coteries of a replica control protocol based on a logical grid network.

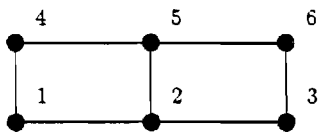


Figure 2: A Grid Network

$$V_{6,3} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \underline{r}_3 = (1,1,1), \quad \underline{w}_3 = (2,2,2)$$

Table 4: Multi-dimensional vote and quorum assignment for the 3x2 grid system

References

[1] L. Lamport, "The implementation of reliable distributed multiprocess systems," *Computer Networks*, vol. 2, pp. 95–114, 1978.

[2] H. Garcia-Molina and D. Barbara, "How to assign votes in a distributed system," *Journal of ACM*, vol. 32, no. 4, pp. 841–860, 1985.

[3] S. Y. Cheung, M. Ahamad, and M. H. Ammar, "Optimizing vote and quorum assignments for reading and writing replicated data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 1, pp. 387 – 397, September 1989.

[4] H. Gifford, "Weighted voting for replicated data," in *Proceedings of 7th Symposium on Operating Systems*, pp. 150 –162, ACM, 1979.

[5] R. H. Thomas, "A majority consensus approach to concurrency control for multiple copy databases," *ACM Transactions on Database Systems*, vol. 4, pp. 180–209, June 1979.

[6] D. Agrawal and A. E. Abbadi, "An efficient solution to the distributed mutual exclusion problem," in *Proceedings of Principles of Distributed Computing*, pp. 193 – 200, ACM, 1989.

[7] M. Maekawa, "A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems," *ACM Transactions on Computer Systems*, vol. 3, pp. 145–159, May 1985.

[8] A. E. Abbadi and S. Toueg, "Maintaining availability in partitioned replicated databases," in *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pp. 240–351, ACM, 1986.

[9] D. Barbara, H. Garcia-Molina, and A. Spauster, "Protocols for dynamic vote reassignment," in *Proceedings of Principles of Distributed Computing*, pp. 195 –205, ACM, 1986.

[10] D. Davcev and W. Burkhard, "Consistency and recovery control for replicated data," in *Proceedings of 10th Symposium on Operating Systems Principles*, pp. 87 – 96, ACM, 1985.

[11] D. Eager and K. Sevcik, "Achieving robustness in distributed database systems," *ACM Transactions on Database Systems*, vol. 8, no. 3, pp. 354–381, 1983.

[12] S. Jajodia and D. Mutchler, "Dynamic voting," in *Proceedings of SIGMOD-87*, pp. 227 – 238, ACM, 1987.

[13] P. Bernstein and N. Goodman, "An algorithm for concurrency control and recovery in replicated distributed databases," *ACM Transactions on Database Systems*, vol. 9, no. 4, pp. 596–615, 1984.

[14] C. Pu, J. Noe, and A. Proudfoot, "Regeneration of replicated objects," in *Proceedings International Conference on Data Engineering*, IEEE, Feb 1986.

[15] J.-F. Paris, "Voting with witnesses: A consistency scheme for replicated files," in *Proceedings of the 6th International Conference on Distributed Computing Systems*, pp. 606 – 612, IEEE, 1986.

[16] D. Agrawal and A. E. Abbadi, "Reducing storage for quorum consensus algorithms," in *Proceedings of Very Large Databases Conference*, pp. 419 – 430, 1988.

[17] S. B. Davidson, H. Garcia-Molina, and D. Skeen, "Consistency in partitioned network," *ACM Computing Survey*, vol. 17, no. 3, pp. 341 – 370, 1985.

[18] A. Kumar and A. Segev, "Optimizing and evaluating algorithms for replicated data concurrency control," in *Proceedings of 9th Symposium on Distributed Computing Systems*, pp. 101 – 109, IEEE, 1989.

[19] D. Barbara and H. Garcia-Molina, "The reliability of voting mechanisms," *IEEE Transactions on Computers*, vol. 36, no. 10, pp. 1197–1208, 1987.

[20] M. Ahamad and M. Ammar, "Performance characterization of quorum-consensus algorithms for replicated data," *IEEE Transactions on Software Engineering*, vol. 15, no. 4, pp. 492 – 496, 1989.

[21] S. Y. Cheung, M. H. Ammar, and M. Ahamad, "The grid protocol: A high performance scheme for maintaining replicated data," in *Proceedings of 6th International Conference on Data Engineering*, pp. 438 – 445, IEEE, 1990.

[22] J.-F. Pâris and D. D. E. Long, "The performance of available copy protocols for the management of replicated data," *To appear in Performance Evaluation*, 1990.

[23] D. D. E. Long and J.-F. Pâris, "Regeneration protocols for replicated objects," in *Proceedings of the 5th International Conference on Data Engineering*, pp. 538–545, 1989.

[24] S. Y. Cheung, M. Ahamad, and M. H. Ammar, "Multi-dimensional voting: A general method for implementing synchronization in distributed systems," Tech. Rep. GIT-ICS-89/35, Georgia Institute of Technology, Atlanta, GA., 1989.

[25] D. Barbara and H. Garcia-Molina, "Mutual exclusion in partitioned distributed systems," *Distributed Computing*, vol. 1, pp. 119 – 132, 1986.

Paper #15

Prototyping a Broadcast Delivery Information System

Mostafa H. Ammar
Hyoung-Joo Kim
Georgia Institute of Technology
GIT-ICS-90/16

May 3, 1990

Abstract

One of the challenges faced by an information system designer is how to configure a low cost system that can support a potentially large user population and still provide good response time. In typical systems the response time increases quickly with load. In this report we consider how to improve the response time performance of an information delivery system by exploiting the inevitable commonality of information need present in a large user base. We use the broadcast delivery of responses in conjunction with a design that allows the response to a user's request to satisfy other requests. We present a preliminary design for a prototype Broadcast Delivery Information System that allows general interactive database access without requiring expensive non-standard hardware. We also discuss how such a system may be configured using equipment already available in the School's Telecommunications Laboratory.

Authors' Address: School of Information and Computer Science Georgia Institute of Technology, Atlanta, GA 30332

Comments on the contents of this preliminary design document are solicited.

1 Introduction

One of the challenges faced by an information system designer is how to configure a low cost system that can support a potentially large user population and still provide good response time. Currently, the typical design strategy is to respond to each user's request by an individual response (i.e., a response directed only to the user originating the request) resulting in an information system where the response time increases quickly with load. In this report we consider how to improve the response time performance of an information delivery system by exploiting the inevitable commonality of information need present in a large user base. Central to our approach is the broadcast of the response to a user's request to all users. The system is designed such that the response to a user's request can satisfy requests made by other users. Considerable effort has been devoted by researchers to the understanding of the performance of broadcast delivery information systems (BDIS). Previous working examples of a BDIS have limited the type of information access to non-interactive read only transactions and/or have limited the format of the data that can be retrieved. Other systems have required user terminals with network interfaces that are capable of decoding incoming data at Gigabit per second rates. Here we focus on the building of a prototype BDIS that allows general interactive database access without requiring expensive non-standard hardware. The target environment is one where multimedia workstations are being used to provide scientific, engineering and business applications for medium to large populations. Through our prototyping efforts we aim to demonstrate that such a system will be inexpensive and capable of providing reasonable quality of service to a large user population.

In section 2 we discuss the basic operation of our proposed BDIS and in section 3 we discuss two examples of experimental BDIS and explain their shortcomings, we also discuss previous work on the performance of BDISs. In section 4 we describe the proposed prototype architecture, hardware and software. Section 5 contains a discussion of some of the variable elements of our design. Section 6 summarizes the contents of this proposal.

2 Basic BDIS Architecture and Protocols

The architecture of a BDIS is shown in figure 1. The system consists of *user terminals* connected to a *service computer* via a communication network. The network is capable of providing efficient broadcast communication from the service computer to the user terminals (e.g., a satellite connection or an Ethernet). The network is also capable of transmitting data from the user terminals to the service computer. The system users submit database queries (e.g., using SQL [6]) using intelligent user terminals (e.g., workstations) that have local storage and processing capabilities. The information stored in the database is subdivided into physical units which we call *pages*. The pages in a database may not be of the same size, and may correspond to other existing subdivisions, e.g., disk blocks. Each query will be processed by the user terminal to which it was submitted in order to determine the minimal set of pages required to satisfy the query. This is done by consulting a local index of the database. The user terminal then issues a set of requests for this set of pages and enters a mode where it is awaiting the receipt of these pages. As the requested pages are received, they are processed to obtain the required response to the user's query.

Page requests are received by the service computer in charge of the database. The requests

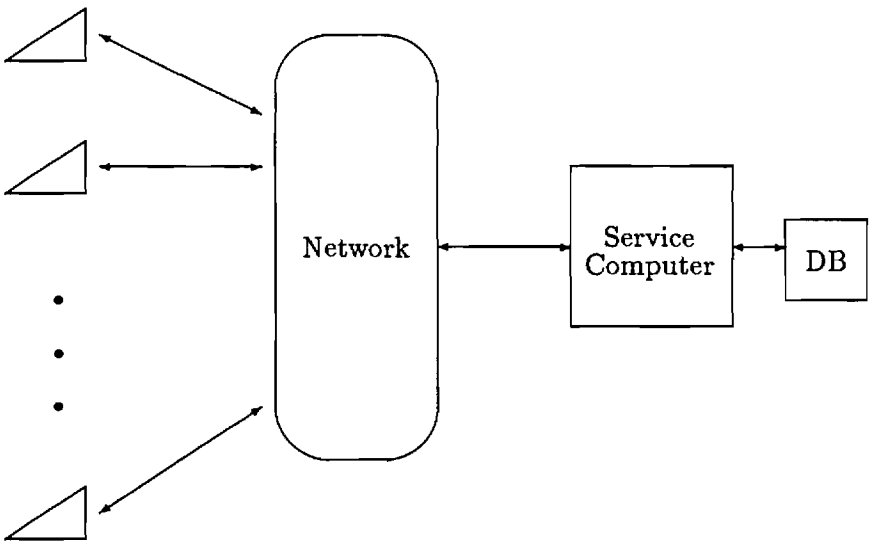


Figure 1: Basic Architecture of a BDIS

arrive in sets corresponding to the pages required to satisfy individual queries. Each page request is treated independently and is preprocessed by a queue manager which decides the order in which the requests are serviced. Different queue disciplines are possible and the choice may have an effect on overall system performance. Service of a page request involves the retrieval of the page from storage and the broadcast of the page to all users.

In a system with a large user population it is expected that unrelated queries will result in page request sets that overlap. Thus, the broadcast of a single page may help in (at least partially) satisfying several outstanding queries simultaneously. This will result in considerable response time reduction, as well as a significant increase in the capacity of the information system.

3 Related Work

3.1 Two Experimental Systems

Two notable examples of broadcast delivery information systems are the Boston Community Information System (BCIS) [10] developed at M.I.T. and the Datacycle architecture [11] developed at Bellcore.

BCIS This system uses an FM broadcast channel to transmit news articles derived from the *New York Times* and the *Associted Press* to users. User terminals are IBM PC's with added software and hardware. The basic mode of operation is the use of simplex (i.e., one-way) broadcast from server computers to the user terminals. Programmable filters at each user terminal capture the desired news articles which are then displayed to the user. Users are allowed to make requests that cannot be satisfied locally, either because they were filtered out or have not been broadcast yet. In such instances a two-way channel is established with the server computers (via dialup modems). This use of simplex broadcast in conjunction with duplex interaction is referred to as *polychannel architecture.*

The major shortcoming of the BCIS is that it assumes a particular information retrieval environment (i.e., news articles) and thus does not allow users to issue updates to the information in the database. Furthermore, the use of information filters requires the users to know the subset of information they may require. This is not reasonable in a system where user terminals may be used by several people and where general database access is desired. Also the BCIS architecture does not allow for efficient updating of the database by the users.

Datacycle Architecture In this system, the contents of the database are continuously transmitted over a high speed (Gigabits per second) broadcast medium. Read requests are satisfied by listening to the broadcast channel and picking up the desired information. Update requests are carried individually over a separate channel to the server computer. Concurrency control is handled through the use of an optimistic scheme.

The major difficulty with the Datacycle architecture is the requirement for user terminal interfaces capable of handling Gigabit per second input rates. Such interfaces can be expensive to manufacture and cannot be bought off-the-shelf. The high data rate is essential because of the

requirement that the entire database be transmitted in a relatively short period of time.

The two above examples have drawbacks that make them a poor fit for environments where users require general database access. In actuality, the users of the system may make high level requests that get translated by interface (or visualization) software into a database query. It is thus essential that the information system (unlike the BCIS) be capable of handling such queries. Furthermore, it is not reasonable to expect that all user workstations are equipped with sophisticated expensive network interfaces such as those required by the DataCycle architecture.

Our prototype design (described in section 4) has the advantage that it can handle general database queries and requires inexpensive off-the-shelf hardware.

3.2 The Performance of Broadcast Delivery

Previous work has investigated the performance of broadcast delivery information systems. This work is summarized below. For details of this work the reader is referred to the papers cited below.

One-Way Systems Investigation of the scheduling of broadcasts in a system with one-way broadcast from the service computer to the users is reported in [4],[3], [2]. The work provides transmission schedules that can drastically improve the response time experienced by users. The response time experienced by the entire user population as well as an individual user are analyzed.

Two-Way Systems The system being proposed fits in this category. In these systems users submit their requests to the service computer. Responses are broadcast to the users. An investigation of the performance gain achieved (over individual response systems) by such systems is reported in [17], [16] ,[8]. Figure 2 shows a comparison of the response time characteristics of an individual response system and a broadcast delivery system (see [17]).

4 Proposed Prototype Architecture

Our objective is to be able to develop a working prototype in a relatively short period of time in order to demonstrate the feasibility and benefit of our proposed approach. To that end, we propose to mostly use equipment that is already available to us in the department's telecommunication laboratory. This laboratory contains a variety of networking hardware and software, including two workstations that can be used as our user terminals. Two additional workstations that are to be used for software development and testing are also needed so that a non-trivial system prototype can be built.

The architecture of our proposed prototype is shown in figure 3. It consists of two baseband Ethernets (A and B) running at 10Mbps each. The Ethernets are already available in our laboratory. A Sun SPARC workstation with 766Mbytes of disk storage (already available in the laboratory) will function as the service computer and will store the database information. Three other Sun SPARC workstations will function as user terminals. It is our intention to simulate a very large user population using these three workstations. All workstations will be connected to both Ethernets. The user terminals will submit their requests over Ethernet B and receive their responses over

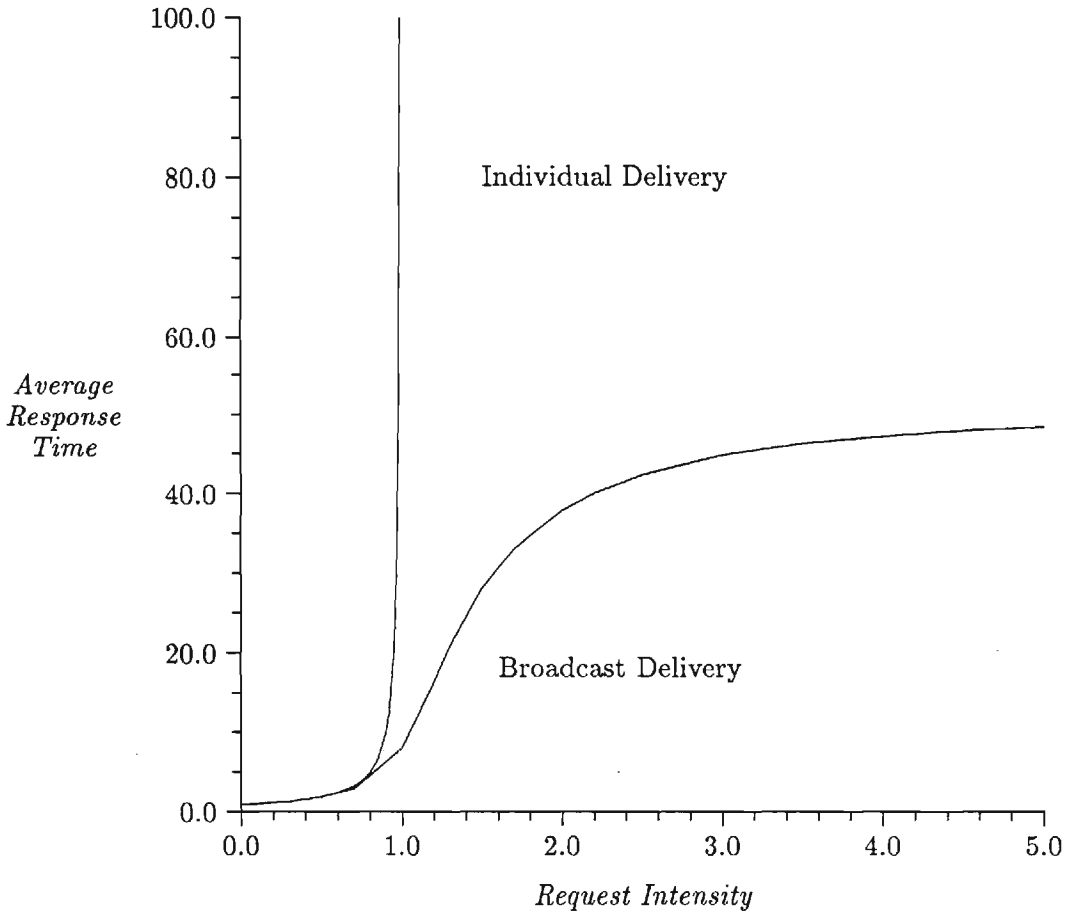


Figure 2: Response Time Performance of a BDIS

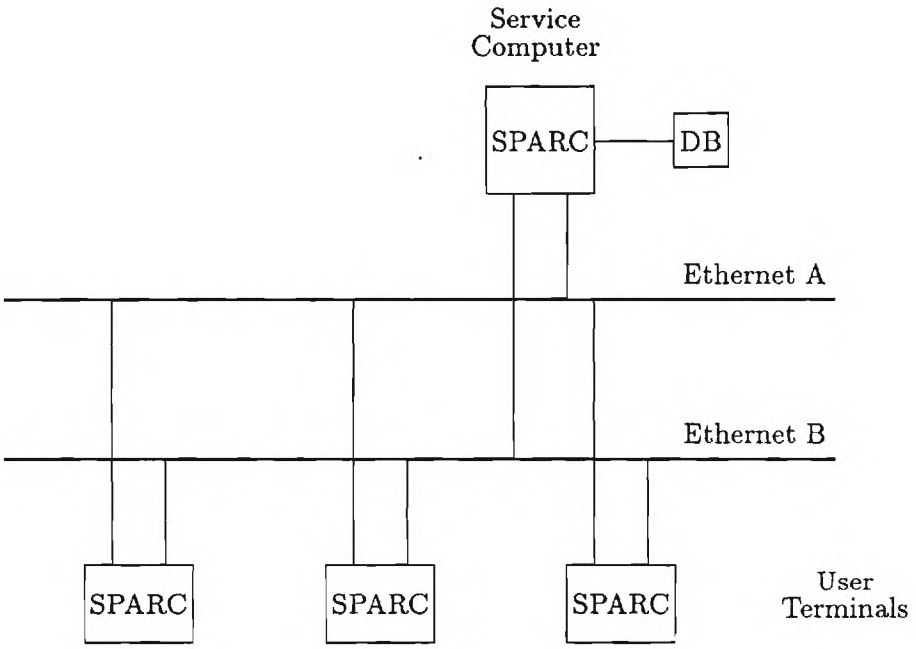


Figure 3: Architecture of Prototype

Ethernet A. Only the service computer workstation will be allowed to transmit on Ethernet A. Contention over Ethernet B will be dealt with as part of the Ethernet protocol.

The user terminal network interfaces are such that they can be programmed to accept messages addressed to a variety of addresses (in addition to the node's own address and the broadcast address). When pages are transmitted, the destination address will contain an identification of the page rather than a node's address. After a user terminal issues page requests, it will instruct its network interfaces to add the page identifiers to the list of accepted addresses. Since address recognition is done in hardware, this approach will save user-terminal workstations from having to decode all incoming messages using the workstation's CPU.

Each user-terminal workstation will have a copy of the index to the database. A central copy of the index will be maintained at the service computer The index will be broadcast periodically (just like an ordinary page) so that the user terminals may maintain an up-to-date copy. Resident software in the workstation will be responsible for:

- **Page Translation:** This is the job of determining, through the use of the local index, the pages required to answer the given query.
- **Query Processing:** This task involves the retrieval of the desired information from the pages received.

We plan to use the Wisconsin Storage System (WiSS) which we already have to achieve the above tasks. WiSS has been designed as a flexible testbed for experimental database systems and especially suitable to provide database services in network environments. (See the appendix for a more detailed description of WiSS.)

The Service computer's software will need to deal with management of the queue of incoming page requests, page retrieval from disk and page transmission. This software will need to be written from scratch. The different queue management techniques are described in more detail in section 5.

5 Design Variables

5.1 Implementing a Read-Only System

In a read-only system some of the variables that need to be considered are listed below. For each we discuss the different options available. For some, the approach we will take in the initial implementation is clear and is motivated by an option's simplicity or by its performance advantage as indicated by previous research. For some others, the option is not clear and will require some studying.

It should be emphasized that our system is intended as an experimental system in which all of the available options may be tested at some point in the future although only one is initially implemented.

Indexing There are a large number of index schemes (single-key schemes to multi-key schemes) in the database literature. During the 70's, only single-key based indexing schemes (such as index-sequential file, B-tree) [5],[9] were studied. As database technology evolved, many sophisticated file structures which provide multikey access to records by combinations of more than one attribute have been proposed in the 80's.

Traditionally, multikey access could be achieved by having several single-key index files (namely, inverted files). Inverted files have several drawbacks, particularly for for multikey access to highly dynamic files (i.e., frequent insertions and deletions). The major drawbacks are: (1) an excessive number of disk accesses to retrieve the inverted files and (2) the insertion and deletion overhead in terms of space and time. Such drawbacks have stimulated the development of many alternative structures that treat all significant attributes symmetrically (i.e., avoid the distinction between the primary and secondary keys). It becomes even more appealing when several attributes are equally significant. A variety of multikey file structures [14],[13] have been proposed to obtain performance better than an inverted file in at least some environments. Each has its strength and its weakness, and also circumstances for which it is well suited.

By analyzing the performance of the different indexing schemes, we plan to identify the promising one from among the proposed single-key or multi-key schemes for the BDIS environment. We will investigate how system's performance is going to be influenced by the choice of indexing scheme, how much user terminal memory is required to store the index, and how fast page translation is for each indexing scheme.

Logical Database Subdivision A user's query gets translated into a set of page requests. The question that needs to be addressed here is how does the choice of database subdivision into pages influence the performance of our scheme. Small pages require a relatively short amount of time to transmit and require less memory to store at the user's terminal. On the other hand, larger pages will offer a greater potential for shared response and thus a lower response time. Our pages will likely be multiples of some natural physical (e.g., disk block) or logical (e.g., file or part of file) database subdivision. We plan to choose an approach to database subdivision via the use of a computer simulation. It should be noted that the choice of database subdivision maybe strongly influenced by the indexing scheme used.

Caching It may be possible to improve system response time by having a user terminal receive pages from the broadcast stream in anticipation of a user's future request. It is also possible to have a terminal hold on to previously received pages in case they are requested again. Request anticipation maybe accomplished via the use of the *linked pages* scheme [2], in which each page contains information about the other pages that are related and are likely to be requested next by the same user.

Caching will require extra memory at each user terminal and extra software development. Its effect on performance is almost obvious: some form of caching will always improve performance. We plan to initially implement our prototype with no caching capabilities at the user terminals. Our judgement of the performance merits of our system will thus be based only on the gains derived from broadcast delivery.

Request Channel Contention In our proposed prototype, the user terminals share a request channel. The method by which contention for this channel is resolved may affect the system's performance. Our initial prototype will use the standard Ethernet (CSMA/CD) protocol. This is attractive mainly because the workstations are already equipped with Ethernet drivers. Other approaches to sharing the channel are possible. For example, in *broadcast polling* [1] the service computer polls the user terminals in groups to determine whether or not a particular page transmission is required. This and other techniques will be considered. However, their implementation may require a considerable effort in the modification of network interface hardware and software.

Queue Management Different methods for queue management at the service computer have been proposed [8]. We propose to use the First-Come, First-Served approach with request discarding in our initial implementation. In this scheme requests for pages are admitted to the queue only if they do not duplicate a page request already awaiting processing. Admitted requests are queued on a FCFS basis. This technique has been shown to have good response time properties as well as providing for a certain amount of fairness. Other techniques [8] that have been proposed include the *Most Requested First* policy where pages are transmitted based on the number of pending requests for each and the *Longest Wait First* policy where pages are transmitted based on the waiting time experience of requesting users.

5.2 Incorporating Data Updates

In a BDIS where update, as well as read transactions are allowed, several new problems are encountered. These problems are listed below. We plan to investigate some potential solutions which will allow us to complete the design of such a system with an eye towards a future implementation as an extension to the read-only system described previously.

Concurrency Control If we allow multiple users to update a database, we need to have a concurrency control scheme for maintaining consistency of database. The *shared response* feature of a BDIS makes concurrency control difficult because responding to several read requests simultaneously may violate the rules of serializability. One simple minded solution to this problem is to have update transactions propagate as is (i.e., with no page translation) to the service computer where they are processed using a conventional concurrency control scheme. Results of processing update transactions are transmitted (individually) to the requesting user. This simple approach will work when the proportion of read-only transactions is high but will suffer performance degradation in environments with many update tranasctions.

Cache Coherence As discussed in the previous subsection, workstations may be allowed to cache information retrieved previously or anticipate users requests by capturing pages from the broadcast stream. When update transactions are allowed, any information cached locally may be out of date and thus should not be used. This problem is very similar to the one encountered in multiprocessor systems with caches. Many cache coherence schemes for multiprocessor systems have been proposed [7],[12],[15], we intend to use or modify the proposed schemes for our BDIS environment.

Index Update Update transactions present the added complexity of how to update the index to the database. This is an especially hard problem in the BDIS environment as the database index is replicated on all workstations and is an integral part of the system operation. The problem is how to reflect any required changes to the index in all the copies. Indices have to be updated in accordance with database updates. In a BDIS, every workstation has its own local index. Local indices should be consistent with the index in the host computer. There are two orthogonal issues in updating indices: (1) who will do the updates? and (2) when will the updates be made?.

Who will do the updates?: There are two schemes. The first scheme (we call it "autonomous update" (AU)) is that the host computer broadcasts updates (not index) and index updates are done by local workstations autonomously. The second scheme (we call it "global update" (GU)) is that the host computer broadcasts the whole newly updated index to local workstations. There is a tradeoff between communication cost (for broadcasting the whole index) and local processing cost (for local index updates).

When will the updates be made?: This problem has been well studied in the context of traditional databases. There are two methods: "immediate update" (IU) and "deferred update" (DU). IU does index update immediately, whereas DU uses a separate data structure, called the *differential file* for keeping database updates temporarily, and then does updates in a batch when the system load is not high. It has been known that the performance of DU is in general better than that of

IU. However, DU incurs a space overhead for the differential file and a search overhead because every index search should accompany a search on differential file for finding recent updates in the differential file.

Since the above two issues are orthogonal, we can think of 4 different combinations in updating indices: AU + IU, AU + DU, GU + IU, and GU + DU. We plan to investigate the pros, cons, and performance of those 4 combinations.

6 Summary

In this technical report we have described our concept of a Broadcast Delivery Information System capable of providing general database access. The system provides for response time improvements over a traditional, individual delivery system by exploiting the common information needs present in a large user population. We have also outlined our design for a read-only system and our approach to extending the system capabilities to allow for update operations. Our focus is on the building of a prototype system using existing hardware and software.

Many extensions to the proposed work are possible. Ongoing work is exploring the use of different types of networks on the performance of a BDIS. Of particular interest has been the use of the satellite based Very Small Aperture Terminal (VSAT) systems.

Appendix: A Brief Description of WiSS

The Wisconsin Storage System (*WiSS*) has been designed as a flexible testbed for experimental database systems, either to store data in non-traditional applications (e.g. object-oriented databases) or to provide database services in a network environment.

WiSS has been implemented on top of UNIX. However, since performance is one of the main concerns, the UNIX file system is replaced by an extent-based file system on top of a "raw" disk, and UNIX buffering is displaced by *WiSS*'s own buffering. *WiSS* supports sequential files, B-tree indices and hash indices. Files can be accessed directly or via scans. Scans can be associated with ordering and conditions, which allows the examination of files in a highly selective manner.

WiSS storage consists of volumes, which contain files, which in turn contain records. Two types of files are provided: *sequential files* (sets of records, each with a *record id*) and *unstructured files* (arrays of bytes). Records can grow and shrink in place, and in general have any length, up to one page.

Sequential files may have indices associated. An *index* maps field values into a collection of records with fields of that value. Hence, indices provide *associative views* of files. Two types of indices are provided: *B+-tree* and *hash*. It is possible to distinguish a file such that the corresponding file is physically ordered in the same manner as its index's key ordering. However, indices must be created and maintained explicitly by the applications: *WiSS* doesn't maintain them.

Sequential files and their indices can be accessed randomly (by record id) or through scans. A *scan* is a method to examine in a given order all the records in a file that match a given *search criterion*. In some cases, record id's or record key ranges can also be specified. Thus, a scan introduces an order on the records of a file. Note that several scans may coexist at a time on the same file.

WiSS also allows the storage of data items longer than a page (the size limit for records). *Long data items* are provided for this purpose. Long data items can be arbitrarily long, and they can be accessed and modified in place. Long data items are internally implemented as sets of shorter records. So, compression is sometimes necessary.

WiSS architecture consists of four layers:

** Level 0 (physical I/O): manages physical disk devices and schedules all I/O activities. It allocates extents to files and manages free space on volumes.

** Level 1 (buffer manager): maintains a buffer pool of pages and implements a place replacement strategy.

** Level 2 (storage structure): implements the record storage abstraction. Sequential files, primary and secondary indices, and long data items are provided.

** Level 3 (access methods): implements the access methods via scans on sequential files, indices, and long data items.

Concurrency control is achieved through locking. The locking granularity is the file.

References

[1] Ammar, M. H., "Teletext-Like Information Delivery Systems Using Broadcast Polling," Computer Networks and ISDN Systems, Vol. 12, # 2, March 1987, pp107-115.

[2] Ammar, M. H., "Response time in a Teletext System; An Individual User's Perspective," IEEE Transactions on Communications, Vol. 35, # 11, November 1987, pp 1159-1170.

[3] Ammar, M. H., Wong, J. W., "On the Optimality of Cyclic Transmission in Teletext Systems," IEEE Transactions on Communications, Vol. 35, # 1, January 1987, pp68-73.

[4] Ammar, M. H., Wong, J. W., "The Design of Teletext Broadcast Cycles," Performance Evaluation, Vol. 5, # 4, November 1985, pp235-242.

[5] Batory, D.S. and C.C. Gotlieb, "A Unifying model of Physical Databases," ACM Trans. on Database Systems, Vol. 7, No. 4, 1982.

[6] Chamberlin, D. D., "A Summary of User Experience with the SQL Data Sublanguage" Proceedings of the INternational Conference on Very Large Data Bases, July 1980, pp 487-503

[7] Cheong, H. and A. Veidenbaum, "A version control approach to cache coherence," *In Proceedings of the 15th Annual International Conference on Supercomputing*, June 1989.

[8] Dykeman, H. D., Ammar, M. H., Wong, J. W., "Scheduling algorithms for Videotex Systems under Broadcast Delivery," Proceedings of the 1986 International Conference on Communications, Toronto, Ontario, June 1986, pp1847-1851.

[9] Fagin, F., Nievergelt, J., Pippenger, N., and Strong, H.R., "Extendible Hashing - A Fast Access Method fro Dynamic Files," ACM Trans. on Database Systems, Vol. 4, No. 3, 1979.

[10] Gifford, D. K., "Polychannel Systems for MAss Digital Communications," Communications of the ACM, Vol. 33, # 2, February 1990, pp141-151.

[11] Herman, G., Gopal, G., Lee K., Weinrib, A., "The Datacycle Architecture for Very High Throughput Database Systems," Proceedings of ACM-SIGMOD, 1987.

[12] Min, S.L. and J.-L. Baer, "A timestamp-based cache coherence scheme," *In Proceedings of the 15th Annual International Conference on Parallel Processing*, Vol. 1 Architecture, August 1989.

[13] Nievergelt, J., Hinterberger, H., and K.C. Sevcik, "The Grid File: An Adaptable, Symmetric Multikey File Structure," ACM Trans. on Database Systems, Vol. 9, No. 1, 1984.

[14] Robinson, J.T., "The k-d-B tree: a search structure for large multidimensional dynamic indexes," ACM SIGMOD, 1981

[15] A.J. Smith, "CPU cache consistency with software support and using one time identifiers," *In Proceedings of the Pacific Computer Communications Symposium*, October 1985.

[16] Wong, J. W., Ammar, M. H., "Response Time Performance of Videotex Systems," IEEE Journal on Selected Areas in Communications, Vol. 4, # 7, October 1986, pp1174-1180.

[17] Wong, J. W., Ammar, M. H., "Analysis of Broadcast Delivery in a Videotex System," IEEE Transactions on Computers, Vol. 34, # 9, September 1985, pp863-866.

Paper #3

T-KDE/1/3//30638

Optimizing Vote and Quorum Assignments for Reading and Writing Replicated Data

Shun Yan Cheung
Mustaque Ahamad
Mostafa H. Ammar

Reprinted from
IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING
Vol. 1, No. 3, September 1989

Optimizing Vote and Quorum Assignments for Reading and Writing Replicated Data

SHUN YAN CHEUNG, MUSTAQUE AHAMAD, AND MOSTAFA H. AMMAR, MEMBER, IEEE

Abstract—**In the weighted voting protocol which is used to maintain the consistency of replicated data, the availability of the data to read and write operations not only depends on the availability of the nodes storing the data but also on the vote and quorum assignments used. We consider the problem of determining the vote and quorum assignments that yield the best performance in a distributed system where node availabilities can be different and the mix of the read and write operations is arbitrary. The optimal vote and quorum assignments depend not only on the system parameters such as node availability and operation mix, but also on the performance measure. We present an enumeration algorithm that can be used to find the vote and quorum assignments that need to be considered for achieving optimal performance. When the performance measure is data availability, an analytical method is derived to evaluate it for any vote and quorum assignment. This method and the enumeration algorithm are used to find the optimal vote and quorum assignment for several systems. The enumeration algorithm can also be used to obtain the optimal performance when other measures are considered.**

Index Terms—**Availability, data replication, fault tolerance, replica control methods, vote and quorum assignment, weighted voting.**

I. Introduction

A DISTRIBUTED system consists of a number of potentially unreliable nodes interconnected via a communication subnetwork. The resources stored at the nodes can be shared and when a node fails, the resources stored at the node become unavailable. Replicating resources at different nodes with independent failure modes can enhance availability and fault tolerance, since a resource could be available even when some nodes have failed. When data are replicated, care must be taken to preserve consistency among the various copies or *replicas*. In addition to increased availability, replication can also provide improved performance of read transactions by reducing the network communication cost since these transactions can access the data from the local replica.

A large number of replica control protocols have been developed to maintain the consistency of replicated data [1]. In this paper, we address the issue of optimization for a voting-based replica control protocol by deriving a general method for finding the optimal settings for the parameters of the protocol. We consider the voting mechanism because it has proven to be flexible and relatively easy to implement.

Voting has been used for various applications in distributed systems. In [2], Gifford proposed its use for synchronizing read and write operations on replicated files. Each file replica is assigned some number of votes and each operation is required to obtain a predefined quorum of votes to proceed. To ensure that a read operation returns the value installed by the last write operation, the read and write operations must acquire r and w number of votes, respectively, such that $r + w > L$, where L is the total number of votes assigned to all replicas. The values r and w are called the read and write quorum. Generally, $r + w = L + 1$ is used which ensures that each read quorum has a nonempty intersection with each write quorum. Since all replicas need not be updated when a write operation completes, timestamps or version numbers must be used in order to determine the value that is written most recently. When version numbers are used, each write quorum must also intersect with every other write quorum, i.e., $2w > L$ [2].

A number of replica control protocols have been derived from weighted voting. Eager and Sevcik introduced a dynamic scheme based on voting that allows the system to switch between normal and failure modes [3] (which have different values for read and write quorums). The system can also change the quorum assignment in the schemes presented in [4]–[6] and the vote assignment can be changed in the scheme described in [7]. Other protocols based on voting are presented in [8]–[10].

The problem of assigning votes to achieve mutual exclusion is addressed by Garcia-Molina and Barbara in [11]. When the quorum for each operation is a majority of all votes assigned, each operation will have mutually exclusive access to the data. In general, mutual exclusion can be guaranteed by defining a set of groups of nodes [12], called a coterie, such that any two groups in a coterie have a nonempty intersection. When voting is used, the groups of nodes that have a majority of the votes constitute a coterie (there exist coteries that cannot be obtained from any vote assignment [11]). In [11], it is shown that only a finite set of vote assignments need to be considered to get all coteries that can be obtained from vote assignments. Thus, it is not necessary to deal with the unbounded set of possible vote assignments. In another work, the same authors have considered the problem of

Manuscript received September 28, 1988; revised July 11, 1989. This work was supported in part by the National Science Foundation under Grants CCR-8806358 and NCR-8604850.

The authors are with the School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA 30332.

IEEE Log Number 8930638.

selecting the vote assignment that results in the highest system availability for mutual exclusion [13]. For a system where all node availabilities are the same, they derived the optimal vote assignment. In [14], Tong and Kain presented an algorithm for assigning votes that maximizes system availability for mutual exclusion in a system where the availability of the nodes can be different. In a related work [15], the authors evaluate the use of the voting mechanism to manage read and write transactions. For systems where node availabilities are identical, values are derived for the optimal degree of replication and for the optimal read quorum.

We consider the problem of optimizing performance for reading and writing replicated data. Since a direct method such as the one described in [14] does not exist for optimizing performance for reading and writing, our approach first identifies the set of vote assignments that need to be considered and then chooses the optimal one. To this end, we present the following techniques which are the major contribution of this paper.

1) A Technique to Enumerate the Vote and Quorum Assignments That Need to be Considered to Optimize a Given Performance Measure for the General Read/Write Case: The measure of interest may be data availability, communication cost, response time, or a combination of these measures. At the heart of our approach is an efficient algorithm to generate the vote and quorum assignments that need to be considered in the optimization. Such enumeration algorithms have only been considered in the literature for the special case of majority quorums. This paper provides a method to enumerate vote and quorum assignments in the general read/write case when the read and write quorums may be different. Quorums (other than the majority) may be useful in improving performance by reducing the cost of more frequent operations.

2) A Technique for Evaluating Availability: We use system availability to illustrate how the set of vote and quorum assignments can be used to find the optimal assignment for reading and writing replicated data. We present an efficient algorithm to evaluate the availability for a system with different node availabilities and arbitrary vote and quorum assignments. Previous methods for evaluating availability are based on set inclusion/exclusion [16] which cannot be used efficiently for larger systems where data are replicated at over 20 nodes.

The paper is organized as follows. We describe the enumeration algorithm in Section II. Section III presents a model of the system and the analysis that derives a method for finding the availability. Some numerical examples are presented in Section IV and we conclude the paper in Section V.

II. Enumerating Vote Assignable Read Coteries

A. Definitions

Let N be the number of nodes that store replicas of a data item. A data item may correspond to a file or several items may be stored in one file. We number the nodes that store the data from 1 to N. Replica i resides at node i and let $U_N = \{1, 2, \cdots, N\}$ be the universe of the replicas. There are two types of operations allowed on the replicas, read and write, and each operation must acquire the consensus of a number of replicas to proceed. A *read group* G is a subset of U_N and is a minimal group of replicas such that a read operation can proceed if all replicas in the group are available (i.e., the nodes where the replicas are stored have not failed). Thus, failure to acquire the consensus of all replicas in any read group causes the read operation to block or abort. A *read coterie* Q_r is a collection of read groups satisfying the following *noncontainment* property, for any read groups G and H:

$$\forall G, H \in Q_r: G \not\subset H.$$

The noncontainment property is a result of the fact that each group is minimal. For instance, if $G \subset H$, then H is not minimal because even when $i \in H - G$ is removed from it, a read operation can still be completed (all replicas in G are available).

A write operation can proceed only if it can acquire the consensus of replicas that constitute a write group. The *write coterie* Q_w corresponding to a given read coterie Q_r is unique and consists of write groups H that satisfy the noncontainment property and also for each $H \in Q_w$:

$$\forall G \in Q_r: G \cap H \neq \phi.$$

We assume in this paper that timestamping is used to identify the current value. When version numbers are used for this purpose, the intersection of any two write groups must also be nonempty. The results of this paper can be modified to accommodate this case.

In voting [2], a special subset of read/write coteries is used, namely those that can be obtained from some vote assignment, and we will call these read/write coteries *vote assignable*. A vote assignment is a vector $V_N = (v_1, v_2, \cdots, v_N)$ where v_i $(1 \leq i \leq N)$ is a nonnegative integer representing the number of votes assigned to replica i. We define $L(V_N)$ to be the total number of votes assigned to the replicas, or $L(V_N) = \Sigma_{i=1}^{N} v_i$. Let group $G = \{g_1, g_2, \cdots, g_k\}$ be a set of replicas where replica g_1 has the least number of votes and we denote by $v(G)$ the sum of the votes assigned to replicas in G. The group G is a *tight group* of s number of votes if and only if the replicas in G collectively have at least s votes and removal of any replica (in particular the replica g_1 which has the least number of votes) from G leaves a group with less than s votes. In other words,

$$G \text{ is a tight group of } s \text{ votes}$$
$$\Leftrightarrow s \leq v(G) \leq (s - 1) + v_{g1}.$$

A *read* group of r votes is a tight group of r votes. The value r is called the read quorum. A *read coterie* with quorum r consists of all the read groups of r votes. A vote assignment V_N and a read quorum r uniquely identify a read coterie. However, the same read coterie can be obtained using different vote assignments and/or different read quorums. For example, the read coterie $\{\{1, 2\}, \{3\}\}$ can be obtained from $V_1 = (1, 1, 2)$ and $r = 2$ and

$V_2 = (2, 3, 5)$ and $r = 4$. For each read coterie with a quorum of r votes there exists a write coterie with a quorum of $L + 1 - r$ votes. The write coterie for a given read coterie is unique and we can limit our attention in the enumeration algorithm to only the read coteries.

For a system with N nodes, we use $Q_N(r, V_N)$ to denote the read coterie obtained from the vote assignment V_N when the read quorum is r. For $1 \leq r \leq L(V_N)$, the read coterie is well-defined. If $r > L(V_N)$, then we define $Q_N(r, V_N) = \phi$, i.e., no read group can be formed. For $r \leq 0$, we define $Q_N(r, V_N)$ to be $\{\phi\}$, i.e., a read operation requires no consensus.

A vote assignable read coterie of N nodes corresponds to a vote and quorum assignment V_N and r, respectively. To determine the optimal assignment, we only need to consider the set of (V_N, r) pairs such that they represent all distinct vote assignable read coteries. We denote this universe of read coteries for N replicas by Ω_N.

Two read coteries R and S are *isomorphic* if and only if there is a permutation π of integers $1, \cdots, N$ such that when we replace each i in S by $\pi(i)$, we obtain R. If the vote assignment V_N is permuted and the read quorum r is kept at the same value, the resulting read coterie will be isomorphic to $Q_N(r, V_N)$. (Permuting the vote assignment will, in general, affect the performance in a system with different node availabilities.) A collection of read coteries E_N is an *enumeration* [11] if every read coterie in Ω_N is either in E_N or is isomorphic to one in E_N and no two read coteries in E_N are isomorphic. Note that $E_N \subseteq \Omega_N$ and E_N can be obtained from Ω_N by choosing one representative from each isomorphic class. Conversely, Ω_N can be obtained from E_N by applying all possible permutations of $1, 2, \cdots, N$ to the members of E_N. In general, Ω_N is the space that must be searched to find the best vote assignable read coterie for a given performance measure.

A summary of the terms used and the notation is presented in Table I.

TABLE I
SUMMARY OF TERMINOLOGY AND NOTATION

Terminology	Notation	Description
Universe of Replicas	U_N	Set of all N replicas
Vote assignment	$\underline{V}_N$	v_i = number of votes assigned to node i
Total Number of Votes	$L(\underline{V}_N)$	$\sum_{i=1}^{N} v_i$
Tight group of s votes		Group that has at least s number of votes and removal of any member leaves a group with less than s votes
Read/write quorum	r/w	Threshold of votes for reading/writing ($r + w = L(\underline{V}_N) + 1$)
Read group of r votes	$G, H, \ldots$	Tight group of r votes
Read Coterie	$Q_N(r, \underline{V}_N)$	Set of all read groups of r votes
Write group of w votes	$G, H, \ldots$	Tight group of w votes
Write Coterie	$Q_N(w, \underline{V}_N)$	Set of all write groups of w votes
Universe of Read Coteries	Ω_N	Set of all read coteries of N copies
Enumeration of Read Coteries	E_N	Set of all non-isomorphic read coteries of N copies

B. Generating All Vote Assignable Read Coteries

We now present an algorithm that can be used to generate Ω_{N+1} when Ω_N is given. Since $\Omega_1 = \{\{\phi\}, \{\{1\}\}, \phi\}$, the algorithm can be used, in principle, to find Ω_N, for any value of N. The algorithm is derived from the results of the following lemma that states how the read coterie of a system with N replicas changes when the system is expanded by creating a new replica (replica $N + 1$) which is assigned v_{N+1} votes.

Lemma 2.1—Expanding the Vote Assignment: Let $V_N = (v_1, v_2, \cdots, v_N)$ be a vote assignment to a system of N replicas and let V_{N+1} be the vote assignment that results when replica $N + 1$ having v_{N+1} votes is added to the system (replicas 1 to N are assigned the same number of votes in both V_N and V_{N+1}). Then,

$$Q_{N+1}(r, V_{N+1}) = Q_N(r, V_N) \cup \{G \cup \{N+1\} \mid G \in G_N(r - v_{N+1}, V_N) \wedge G \notin Q_N(r, V_N)\}.$$

Proof: See Appendix A.

We know from Lemma 2.1 that for each vote assignable read coterie $Q_{N+1}(r, V_{N+1})$ of $N + 1$ replicas there must exist read coteries Q_1 and Q_2 of N replicas ($Q_1 = Q_N(r, V_N)$ and $Q_2 = Q_N(r - v_{N+1}, V_N)$) such that $Q_{N+1}(r, V_{N+1})$ is related to Q_1 and Q_2 as stated in the lemma. The algorithm presented in Fig. 1 uses this fact as it generates read coteries of a system of $N + 1$ replicas by combining every pair of the read coteries of N replicas using the relationship defined by Lemma 2.1. Notice that since Q_1 and Q_2 in Lemma 2.1 have the same vote assignment and we are combining all pairs (even those which are obtained from different vote assignments), the algorithm has to check that the resulting set can be obtained by some vote assignment. The output of the algorithm, Ω, will be Ω_{N+1}. This follows from the fact that each read coterie $Q \in \Omega_{N+1}$ can be written as a combination of some pair of read coteries in Ω_N (as given by Lemma 2.1) and every possible pair of read coteries of Ω_N is combined by the algorithm in the manner given by Lemma 2.1, hence, Q will be generated.

Note that Q, the set generated by the statement (†) of the algorithm, may not satisfy the noncontainment property for read coteries. For example, $Q_1 = \{\{1\}\}$, $Q_2 = \{\{1, 2\}\}$ and $N + 1 = 3$, the set $Q = \{\{1\}, \{1, 2, 3\}\}$ is generated which violates the noncontainment property. However, such sets are not vote assignable and will not be included in Ω. The statement (†) also generates read coteries (satisfying the noncontainment property) that are not vote assignable.

Determining whether the set Q is vote assignable can be done by formulating a linear program (LP) from the groups of Q. We define the following set of groups of $N + 1$ replicas:

$$Y(Q) = \{H \in 2^{U_{N+1}} \mid H \text{ is not a superset of any group of } Q \text{ or } H \text{ is a proper subset of a group of } Q\}$$

where $2^{U_{N+1}}$ is the set of all subsets of the universe of $N + 1$ nodes. When a group G is in Q, any proper subset of

TABLE II
READ COTERIES OF FOUR NODES

Index	Read Set	Vote assignment				
		v_1	v_2	v_3	v_4	r
1	{{4}}	0	0	0	1	1
2	{{3}, {4}}	0	0	1	1	1
3	{{34}}	0	0	1	1	2
4	{{2}, {3}, {4}}	0	1	1	1	1
5	{{23}, {24}, {34}}	0	1	1	1	2
6	{{234}}	0	1	1	1	3
7	{{1}, {23}}	0	1	1	2	2
8	{{21}, {31}}	0	1	1	2	3
9	{{1}, {2}, {3}, {4}}	1	1	1	1	1
10	{{12}, {13}, {14}, {23}, {24}, {34}}	1	1	1	1	2
11	{{123}, {124}, {134}, {234}}	1	1	1	1	3
12	{{1234}}	1	1	1	1	4
13	{{4}, {12}, {13}, {23}}	1	1	1	2	2
14	{{14}, {24}, {34}, {123}}	1	1	1	2	3
15	{{124}, {134}, {234}}	1	1	1	2	4
16	{{4}, {123}}	1	1	1	3	3
17	{{14}, {24}, {34}}	1	1	1	3	4
18	{{3}, {1}, {12}}	1	1	2	2	2
19	{{13}, {14}, {23}, {24}, {34}}	1	1	2	2	3
20	{{34}, {123}, {124}}	1	1	2	2	4
21	{{134}, {234}}	1	1	2	2	5
22	{{4}, {13}, {23}}	1	1	2	3	3
23	{{34}, {124}}	1	1	2	3	5
24	{{14}, {23}, {24}, {34}}	1	2	2	3	4
25	{{24}, {34}, {123}}	1	2	2	3	5

For a given N, since Ω_N can be obtained from E_N, optimal vote and quorum assignments can be determined for any particular performance measure by an exhaustive search. In what follows we illustrate this by considering the optimization of the availability of replicated data for read and write operations. Although there is a direct method for obtaining the settings for voting that maximizes the system availability for mutual exclusion [14], no direct method is known for finding optimal settings for reading and writing.

III. PERFORMANCE ANALYSIS OF WEIGHTED VOTING

We now develop a method for evaluating the availability of replicated data for a given vote assignment and read quorum when the system consists of N nodes and each node stores a replica of the data item. This method will be used to evaluate the performance of the vote and quorum assignments enumerated in the previous section in order to determine the best one.

For each node i, $i = 1, \cdots, N$, we assume that the mean time-to-fail is $1/\lambda_i$ and the mean time-to-repair is $1/\mu_i$. We also define the parameter p_i which represents the proportion of time the node is operational. We thus have

$$p_i = \frac{\frac{1}{\lambda_i}}{\frac{1}{\lambda_i} + \frac{1}{\mu_i}} = \frac{\mu_i}{\lambda_i + \mu_i}. \quad (1)$$

The parameter p_i will be called the *availability* of node i and can be viewed as the steady-state probability that the node is operational.[2] We will use the *availability vector* $\boldsymbol{P}$ to denote $(p_1, \cdots, p_N)$. The vote assignment under consideration is denoted by $\boldsymbol{V} = (v_1, \cdots, v_N)$ and we define $L = \sum_{i=1}^{N} v_i$ (we drop the subscript N from our earlier notation for simplicity).

The availability for operations requiring s votes is the proportion of time (or steady-state probability) that the total number of votes from all operational nodes is equal to or exceeds s. We denote this probability for given quorum s, availability vector $\boldsymbol{P}$, and vote assignment $\boldsymbol{V}$ by $\alpha_s(\boldsymbol{P}, \boldsymbol{V})$ ($s = r$ for read access, $s = w$ for write access, and $r + w = L + 1$). We use the system availability α as the performance measure in the analysis. The system availability for read/write transactions is equal to $\alpha(\boldsymbol{P}, \boldsymbol{V}) = f\alpha_r(\boldsymbol{P}, \boldsymbol{V}) + (1 - f)\alpha_w(\boldsymbol{P}, \boldsymbol{V})$, with f being the fraction of read operations.

In order to derive an expression for $\alpha_s(\boldsymbol{P}, \boldsymbol{V})$ for given values of s, $\boldsymbol{P}$, and $\boldsymbol{V}$, we define the state of the system $\boldsymbol{n} = (n_1, \cdots, n_N)$ where $n_i = 1$ if node i is operational and $n_i = 0$ otherwise. Let $P(\boldsymbol{n})$ be the steady-state probability that the system is in state $\boldsymbol{n}$. Observe that $P(\boldsymbol{n})$ is the probability that all nodes with $n_i = 1$ are operational and that all nodes with $n_i = 0$ have failed. Thus, we have

$$P(\boldsymbol{n}) = \prod_{i=1}^{N} (1 - p_i)^{(1-n_i)} p_i^{n_i} = \left[\prod_{i=1}^{N} (1 - p_i)\right] \prod_{i=1}^{N} \left(\frac{p_i}{1 - p_i}\right)^{n_i}. \quad (2)$$

We next obtain an expression for $Q(m)$, the steady-state probability that the total number of votes available in the system is m. This is given by the sum of all state probabilities in (2) such that $C(\boldsymbol{n}, \boldsymbol{V}) = \sum_{i=1}^{N} n_i v_i = m$. Thus, we get for $m = 0, 1, 2, \cdots, L$

$$Q(m) = \sum_{\text{all } \boldsymbol{n} \text{ s.t. } C(\boldsymbol{n}, \boldsymbol{V}) = m} P(\boldsymbol{n}) = \frac{1}{K} h(\boldsymbol{V}, m) \quad (3)$$

where we define

$$\frac{1}{K} = \prod_{i=1}^{N} (1 - p_i) \quad (4)$$

$$h(\boldsymbol{V}, m) = \sum_{\text{all } \boldsymbol{n} \text{ s.t. } C(\boldsymbol{n}, \boldsymbol{V}) = m} \prod_{i=1}^{N} q_i^{n_i} \quad (5)$$

and $q_i = p_i/(1 - p_i)$. Since the operation requires s votes, we obtain that

$$\alpha_s(\boldsymbol{P}, \boldsymbol{V}) = \sum_{m=s}^{L} Q(m) = \frac{1}{K} \sum_{m=s}^{L} h(\boldsymbol{V}, m). \quad (6)$$

[2] An operation can proceed to completion if it can find a required quorum within a certain time-out period. There may be several reasons why nodes are unable to reply within the given time-out period. A node can suffer from a hardware failure or it may be operational but isolated from the other nodes because of network failures. Also, long delays due to network congestion and slow response times due to overload can cause an operation to abort because of time-out. A node that is unable to respond within the time-out period is said to have failed.

```
Ω := φ; (Ω is the output of the algorithm)
for every Q1 ∈ ΩN do
    for every Q2 ∈ ΩN do
        Q := Q1 ∪ {G ∪ {N + 1} | G ∈ Q2 ∧ G ∉ Q1}; ....(†)
        if Q is obtainable from some (V_{N+1}, r) and Q ∉ Ω then
            Record (V_{N+1}, r);
            Ω := Ω ∪ Q;
```

Fig. 1. Algorithm 1.

G must have less than r votes. Also when a group H does not contain any group of Q, it cannot have r or more votes (if it did, then H or its subset must be in Q). The set $Y(Q)$ is thus the set of all groups that do not have the required quorum of r votes. We use this property of $Y(Q)$, coupled with the fact that all the groups in Q must have at least r votes, to formulate the following LP:

$$\min \sum_{i=1}^{N+1} v_i + r$$

$$\text{s.t.: } \forall G \in Q: \sum_{g \in G} v_g \geq r$$

$$\forall H \in Y(Q): \sum_{h \in H} v_h \leq r - 1$$

$$v_i \geq 0,\ i = 1, 2, \cdots, N+1$$

$$r \geq 1.$$

If the LP does not have a solution, then there are no values V_{N+1} and r that satisfy the constraints. If the LP does produce a solution, the values of V_{N+1} and r are rational and since multiplying V_{N+1} and r by the same value will not affect the resulting read coterie, we can always convert the solution to an integer vote assignment and quorum. In principle, we are only concerned with obtaining a feasible solution to the LP. However, the complexity of the performance analysis method described in Section III is a function of the votes v_i and quorum r. We are thus using the indicated objective function.

C. Generating An Enumeration of Vote Assignable Read Coteries

The complexity of Algorithm 1 depends on the size of the input set Ω_N. Also note that, for a given N, we only need to generate the enumeration set E_N from which Ω_N can be obtained. Due to the fact that, in general, E_N is much smaller than Ω_N (e.g., $|E_5| = 119$ and $|\Omega_5| = 3287$), an algorithm which generates E_{N+1} when E_N is given as input, will require less CPU time. Such an algorithm can be derived from Algorithm 1 by only including a single member from a class of isomorphic read coteries. The following lemma provides a simple technique to achieve this and the resulting enumeration algorithm is shown in Fig. 2.

Lemma 2.2—Equality of Isomorphic Read Coteries of Nondecreasing Vote Assignments: Let $V_N = (v_1, v_2, \cdots, v_N)$ and $W_N = (w_1, w_2, \cdots, w_N)$ be two nondecreasing vote assignments, i.e., $v_1 \leq v_2 \leq \cdots \leq v_N$ and $w_1 \leq w_2 \leq \cdots \leq w_N$. $Q_N(r, V_N)$ and $Q_N(s, W_N)$ are isomorphic if and only if $Q_N(r, V_N) = Q_N(s, W_N)$.

```
E := φ; (E is the output of the algorithm)
for every Q1 ∈ EN do
    for every Q2 ∈ EN do
        Q := Q1 ∪ {G ∪ {N + 1} | G ∈ Q2 ∧ G ∉ Q1};
        if Q is obtainable from some (V_{N+1}, r) then
            V'_{N+1} := sort(V_{N+1}); (sort in non-decreasing order)
            if Q_{N+1}(r, V'_{N+1}) ∉ E then
                Record (V'_{N+1}, r);
                E := E ∪ Q_{N+1}(r, V'_{N+1});
```

Fig. 2. Algorithm 2.

Proof: See Appendix A.

The above lemma shows that in each class of isomorphic read coteries, there is *exactly one* member that is obtained from a nondecreasing vote assignment, i.e., two read coteries that are obtained from nondecreasing vote assignments are either equal or nonisomorphic. Then, sorting the vote vector in nondecreasing order in Algorithm 2 will guarantee that $E = E_{N+1}$. The following theorem shows the correctness of Algorithm 2.

Theorem 2.1—Enumeration of Vote Assignable Read Coteries: Let E be the output generated by Algorithm 2, then $E = E_{N+1}$ which is the enumeration of Ω_{N+1} that contains only read coteries that have nondecreasing vote assignments.

Proof:

Claim 1: Every vote assignable read coterie of $N + 1$ replicas is either in E or is isomorphic to a read coterie in E.

Let $Q_{N+1}(r, V_{N+1})$ be an arbitrary read coterie of $N + 1$ replicas and V'_{N+1} be the nondecreasing vote assignment obtained from V_{N+1}. $Q_{N+1}(r, V'_{N+1})$ and $Q_{N+1}(r, V_{N+1})$ are isomorphic and by Lemma 2.1 we can write $Q_{N+1}(r, V'_{N+1})$ as

$$Q_{N+1}(r, V'_{N+1}) = Q_N(r, V'_N) \cup \left\{ G \cup \{N+1\} \mid G \in Q_N(r - v'_{N+1}, V'_N) \wedge G \notin Q_N(r, V'_N) \right\}$$

and since $Q_N(r, V'_N)$ and $Q_N(r - v'_{N+1}, V'_N)$ are in E_N, $Q_{N+1}(r, V'_{N+1})$ is included in E.

Claim 2: There are no isomorphic read coteries in E.

Each vote assignment is sorted in nondecreasing order and by Lemma 2.2 we conclude that no two read coteries in E are isomorphic. □

Table II contains the vote assignments, quorums, and read coteries of E_4 which is produced by repeating the enumeration algorithm three times starting with $E_1 = \{\{\phi\}, \{\{1\}\}, \phi\}$. The read coteries $\{\phi\}$ and ϕ are not included in the table but are elements of E_4. We have also generated E_5, E_6, and E_7. These enumerations have 119, 1113, and 29375 members, respectively.[1]

[1]The enumeration method has so far produced only integral valued vote assignments for every LP generated. It is known that when the constraint matrix of the LP is totally unimodular, the solution is integral valued [17]. The constraint matrices generated by the algorithm do not satisfy this property.

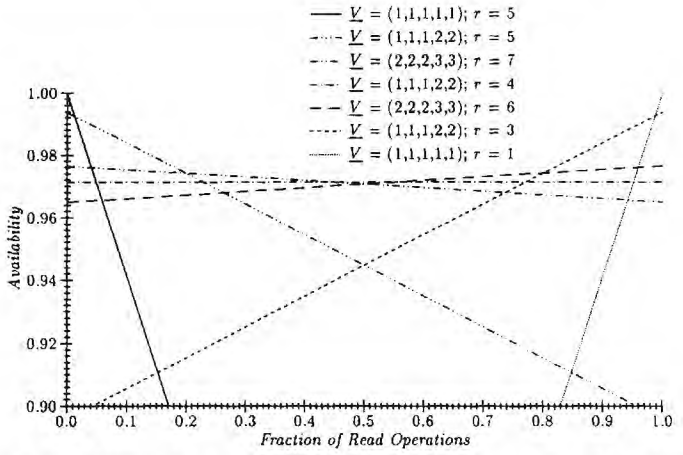


Fig. 4. Read/write availability of read coteries, $p_1 = p_2 = p_3 = 0.8$, other nodes = 0.9.

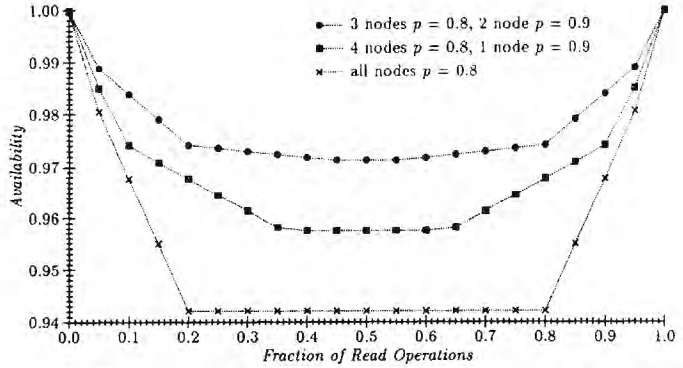


Fig. 5. Optimal read/write availability for three systems of five nodes.

To further illustrate how the optimal vote assignment depends on the node availabilities and f, in Table IV we also consider a system of seven nodes, where the availability of each node is different ($p_1 = 0.65$, $p_2 = 0.7$, $p_3 = 0.75$, $p_4 = 0.8$, $p_5 = 0.85$, $p_6 = 0.9$, and $p_7 = 0.95$). For $f = 0.9$, we see that the most available node is assigned seven votes while the least available node gets only one vote. For $f = 0.8$, the difference in the votes is even more marked.

Finally, to show that the availability obtained from optimal vote and quorum assignment can be appreciably higher than that of the commonly used quorums with a uniform vote assignment (i.e., when $v_i = 1$ for all i), we consider the data in Table V. In the system, two nodes are highly available ($p = 0.9$) while the others only have an availability of 0.6. We see that when $f = 0.8$, the optimal availability is about 9 percent higher compared to the read majority/write majority quorum. Even when the optimal quorum, r_{opt}, is considered for a uniform vote assignment in this system, the availability is still 5.5 percent lower than that achievable with optimal vote and quorum assignment. The read one/write all quorum has poorer availability for all values of f except when f is 0.999. Thus, compared to the commonly used quorums with each node having a single vote, the optimal vote and quorum assignment provides better system availability.

TABLE IV
BEST READ/WRITE SETS FOR A SYSTEM OF SEVEN NODES WITH AVAILABILITY VECTOR = (0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95)

f	Vote assignment	r
0.001	(1, 1, 1, 2, 2, 3, 3)	11
0.1	(1, 2, 2, 3, 4, 5, 7)	15
0.2	(3, 4, 5, 6, 8, 10, 13)	28
0.3	(2, 3, 4, 5, 6, 8, 10)	21
0.4	(2, 2, 3, 4, 5, 6, 8)	16
0.5	(1, 2, 3, 3, 4, 5, 7)	13
0.6	(2, 2, 3, 4, 5, 6, 8)	15
0.7	(2, 3, 4, 5, 6, 8, 10)	18
0.8	(3, 4, 5, 6, 8, 10, 13)	22
0.9	(1, 2, 2, 3, 4, 5, 7)	10
0.999	(1, 1, 1, 2, 2, 3, 3)	3

TABLE V
OPTIMIZING VOTE AND QUORUM VERSUS OPTIMIZING QUORUM USING UNIFORM VOTE ASSIGNMENT FOR A SYSTEM OF SEVEN NODES WITH AVAILABILITY VECTOR (0.6, 0.6, 0.6, 0.6, 0.6, 0.9, 0.9)

$\alpha^{(1)}$ = optimum availability over all vote assignments and quorums
$\alpha^{(2)}$ = availability of the read one/write all quorum
$\alpha^{(3)}$ = availability of the read majority/write majority quorum
$\alpha^{(4)}$ = optimum availability using the uniform vote assignment

	All Vote Assignments			Uniform Vote Assignment			
f	Optimal $\underline{V}_N, r$		$\alpha^{(1)}$	$\alpha^{(2)}$	$\alpha^{(3)}$	r_{opt}	$\alpha^{(4)}$
0.001	(1, 1, 1, 1, 1, 1, 1)	7	0.999	0.064	0.866	7	0.999
0.1	(0, 0, 0, 0, 0, 1, 1)	2	0.972	0.157	0.866	5	0.937
0.2	(1, 1, 1, 1, 1, 5, 5)	10	0.955	0.250	0.866	5	0.901
0.3	(1, 1, 1, 1, 1, 4, 4)	8	0.943	0.344	0.866	4	0.866
0.4	(1, 1, 1, 1, 1, 3, 3)	6	0.933	0.438	0.866	4	0.866
0.5	(1, 1, 1, 1, 1, 3, 3)	6	0.933	0.531	0.866	4	0.866
0.6	(1, 1, 1, 1, 1, 3, 3)	6	0.933	0.625	0.866	4	0.866
0.7	(1, 1, 1, 1, 1, 4, 4)	6	0.943	0.719	0.866	4	0.866
0.8	(1, 1, 1, 1, 1, 5, 5)	6	0.955	0.813	0.866	3	0.901
0.9	(0, 0, 0, 0, 0, 1, 1)	1	0.972	0.906	0.866	3	0.937
0.999	(1, 1, 1, 1, 1, 1, 1)	1	0.999	0.999	0.866	1	0.999

V. CONCLUDING REMARKS

We have presented a method for obtaining an enumeration of vote assignable read coteries and their corresponding vote assignments and quorums. We have also developed an efficient method for finding the availability for any vote assignment when the availability of the nodes and the read/write transaction mix are given. The use of this method was demonstrated by finding the vote assignment and quorum that yields the highest system availability in systems with different node availabilities. It should be emphasized that the enumeration of the vote assignable read coteries can also be used in the optimization of other performance measures. In future research, we plan to study the effect of the communication network and the performance of dynamic voting schemes. We will also consider direct methods for finding optimal settings in voting that maximizes system availability for reading and writing that can be used for optimizing larger systems.

Computing $h(V, m)$ can be accomplished by observing that[3]

$$h(V, m) = \sum_{\text{all } \boldsymbol{n} \text{ s.t. } C(\boldsymbol{n}, V) = m, n_N = 0} \prod_{i=1}^{N} q_i^{n_i} + \sum_{\text{all } \boldsymbol{n} \text{ s.t. } C(\boldsymbol{n}, V) = m, n_N = 1} \prod_{i=1}^{N} q_i^{n_i}$$

$$= \sum_{\text{all } \boldsymbol{n} \text{ s.t. } C(\boldsymbol{n}, V) = m, n_N = 0} \prod_{i=1}^{N} q_i^{n_i} + q_N \sum_{\text{all } \boldsymbol{n} \text{ s.t. } C(\boldsymbol{n}, V) = m - v_N, n_N = 0} \prod_{i=1}^{N} q_i^{n_i}.$$

Alternatively, we can write

$$h(V, m) = h^{-N}(V, m) + q_N h^{-N}(V, m - v_N) \quad (7)$$

where $h^{-N}(V, m)$ is the same as $h(V, m)$ except that it is evaluated for a system with node N removed, i.e.,

$$h^{-N}(V, m) = \sum_{\text{all } \boldsymbol{n} \text{ s.t. } C^{-N}(\boldsymbol{n}, V) = m} \prod_{i=1}^{N-1} q_i^{n_i}$$

where $C^{-N}(\boldsymbol{n}, V) = \sum_{i=1}^{N-1} n_i v_i$.

For a given V, the algorithm given in Fig. 3 that is derived from (7) can be used to evaluate $H(m) = h(V, m)$.

If the vote assignment and read quorum of each read set in Ω_N is given, the algorithm described in Fig. 3 can be used to compute $\alpha(P, V)$ for each (V, r) corresponding to read sets in Ω_N. The (V, r) that yields the highest value for $\alpha(P, V)$ is the optimal vote and quorum assignment. However, if the nodes are labeled such that $p_1 \leq p_2 \leq \cdots \leq p_N$, we need only consider the nondecreasing vote assignments which correspond to the members of E_N generated by Algorithm 2. In other words, the members in $\Omega_N - E_N$ need not be considered for optimizing availability since they are clearly suboptimal. This follows from the intuitive idea that the best vote assignment is the one that assigns more votes to nodes with higher availability which is stated as the following theorem. Note, however, that for a different performance measure, all elements of Ω_N may have to be considered.

Theorem 3.1: Given an availability vector P and a vote assignment V, where, without loss of generality, $p_1 \leq p_2 \leq \cdots \leq p_N$, $v_1 \leq v_2 \leq \cdots \leq v_N$. Then,

$$\alpha(P, V) \geq \alpha(P, V')$$

for any permutation V' of the vote assignment V.

Proof: See Appendix B.

IV. Numerical Examples

We demonstrate the use of the results derived in the previous sections by analyzing systems of five and seven nodes. For these systems, we show the optimal vote and quorum assignment and the resulting system availability.

[3]This is the same technique used to define the basic relationship for the convolution algorithm used to evaluate closed queueing networks [18].

```
Initialize:  H(0) := 1; H(1), H(2), ..., H(L) := 0;
for i := 1 to N do
    for m := L downto v_i do
        H(m) := H(m) + q_i * H(m - v_i)
    end;
end;
```

Fig. 3. Algorithm for computing $h(V, m)$.

TABLE III
Best Read/Write Coteries for Two Systems of Five Nodes

	$p_5 = 0.9$, others $= 0.8$		$p_4 = p_5 = 0.9$, others $= 0.8$	
f	Vote assignment	r	Vote assignment	r
0.001	(1, 1, 1, 1, 1)	5	(1, 1, 1, 1, 1)	5
0.1	(1, 1, 1, 1, 1)	4	(1, 1, 1, 2, 2)	5
0.2	(1, 1, 1, 1, 2)	4	(1, 1, 1, 2, 2)	5
0.3	(1, 1, 1, 1, 2)	4	(2, 2, 2, 3, 3)	7
0.4	(1, 1, 1, 1, 1)	3	(2, 2, 2, 3, 3)	7
0.5	(1, 1, 1, 1, 1)	3	(1, 1, 1, 2, 2)	1
0.6	(1, 1, 1, 1, 1)	3	(2, 2, 2, 3, 3)	6
0.7	(1, 1, 1, 1, 2)	3	(2, 2, 2, 3, 3)	6
0.8	(1, 1, 1, 1, 2)	3	(1, 1, 1, 2, 2)	3
0.9	(1, 1, 1, 1, 1)	2	(1, 1, 1, 2, 2)	3
0.999	(1, 1, 1, 1, 1)	1	(1, 1, 1, 1, 1)	1

Table III shows the optimal vote and quorum assignments for two systems of five nodes for various values of the transaction mix f. Since the system considered in the first column is relatively homogeneous (four nodes have the same availability which is 0.8 and the availability of the other node is 0.9), either the uniform vote assignment is optimal (each replica gets one vote) or the node with higher availability is assigned an extra vote. This does not hold, as shown in column two of the table, when the availability of two nodes is 0.9. In this case, the optimal vote assignment is not uniform for all cases except when f is close to 0 or 1.

We show the system availability in Fig. 4 for the vote and quorum assignments of Table III when the availability of three nodes is 0.8 and it is 0.9 for the other two nodes. Clearly, no single vote and quorum assignment can provide optimal availability for all values of f. For example, (2, 2, 2, 3, 3) with $r = 6$ is optimal for $f = 0.6$ but when f changes to 0.8, the optimal assignment becomes (1, 1, 1, 2, 2) with $r = 3$. Also, note that the availability is not very sensitive to a change in f for some of the vote and quorum assignments.

The optimal system availability as a function of f for the two systems considered above and one where the availability of each node is 0.8, is shown in Fig. 5. The plot for the system of Fig. 5 is obtained from the plots of Fig. 4 by taking the highest system availability for a given f. Although the optimal availabilities of these systems are similar when f is close to 0 or 1, for other values of f, α increases by almost 2 percent when the availability of one node changes from 0.8 to 0.9. Also, the optimal availability for each of the systems is not sensitive to a change in f when f is not close to 0 or 1.

$$\Rightarrow \forall g \in G_1 : v(G_1) + w_a - w_g < s \wedge v(G_1) < s \quad [w_a \leq w_b]$$

$$\Rightarrow \text{if } v(G_1 \cup \{a\}) \geq s, \text{ then } G_1 \cup \{a\} \in Q_N(s, W_N)$$

$$\Rightarrow v(G_1 \cup \{a\}) < s \quad [G_1 \cup \{a\} \notin Q_N(s, W_N)].$$

And,

$$G_1 \cup \{a\} \in Q_N(r, V_N) \wedge G_1 \cup \{b\} \notin Q_N(r, V_N)$$

$$\Rightarrow v(G_1 \cup \{b\}) \geq v(G_1 \cup \{a\}) \geq r \quad [v_b \geq v_a]$$

$$\Rightarrow \text{sub}(G_1 \cup \{b\}) \in Q_N(r, V_N) \quad [G_1 \cup \{b\} \notin Q_N(r, V_N)]$$

$$\Rightarrow \text{sub}(G_1) \cup \{b\} \in Q_N(r, V_N) \quad [G_1 \cup \{a\} \in Q_N(r, V_N)]$$

$$\Rightarrow \text{sub}(G_1) \cup \{a\} \in Q_N(s, W_N)$$

$$\Rightarrow v(\text{sub}(G_1) \cup \{a\}) \geq s$$

$$\Rightarrow v(G_1 \cup \{a\}) > s$$

contradicting the previous conclusion.

The case for $G = G_1 \cup \{b\}$ is similar to the previous one. If $G_1 \cup \{a\} \in Q_N(r, V_N)$ also, then $G_1 \cup \{b\} \in Q_N(s, W_N)$ which contradicts that $G \notin Q_N(s, W_N)$. So it must be that $G_1 \cup \{a\} \notin Q_N(r, V_N)$ and as in the previous case, we can derive contradicting conclusions that $v(G_1 \cup \{a\}) < r$ and $v(G_1 \cup \{a\}) > r$.

Lemma 2.2—Equality of Isomorphic Read Coteries of Nondecreasing Vote Assignments: Let $V_N = (v_1, v_2, \cdots, v_N)$ and $W_N = (w_1, w_2, \cdots, w_N)$ be two nondecreasing vote assignments. $Q_N(r, V_N)$ and $Q_N(s, W_N)$ are isomorphic if and only if $Q_N(r, V_N) = Q_N(s, W_N)$.

Proof: We will use induction to prove that if $Q_N(r, V_N)$ and $Q_N(s, W_N)$ are isomorphic then they are equal. The converse is obvious.

Let π be a permutation that maps $Q_N(r, V_N)$ to $Q_N(s, W_N)$, i.e., $Q_N(s, W_N) = \pi(Q_N(r, V_N))$ where $\pi(Q_N(r, V_N))$ denotes the read coterie obtained by replacing i by $\pi(i)$ in the read coterie $Q_N(r, V_N)$. From the theory of permutations (see for example [19]), we know that a permutation can be factorized into disjoint cycles and this factorization is unique except for the order in which the cycles are written. Also, each l-cycle $(i_1\ i_2\ \cdots\ i_l)$ can be written as a product of $l - 1$ transpositions as follows: $(i_l\ i_1)\ (i_{l-1}\ i_1)\ \cdots\ (i_2\ i_1)$. Therefore, if a permutation π can be factorized into c cycles and cycle i is of length l_i, then π can be factorized into $\Sigma_{i=1}^{c}(l_i - 1)$ transpositions. The proof of the lemma is by induction on the number of transpositions that constitute the factorization of π. For clarity, we use τ to denote a transposition.

Basis: $\pi = \tau_1 = (a\ b)$.

Without loss of generality, assume $a < b$. Then,

$$Q_N(s, W_N) = \tau_1(Q_N(r, V_N)).$$

We thus have by Lemma A.3 that $Q_N(s, W_N) = Q_N(r, V_N)$ and the basis is proved.

Induction Hypothesis: Given that V_N and W_N are nondecreasing, if $Q_N(r, V_N)$ and $Q_N(s, W_N)$ are isomorphic under a permutation π_k that has a factorization of k or less transpositions, then $Q_N(r, V_N)$ and $Q_N(s, W_N)$ are equal.

Induction Step: We need to show that when V_N and W_N are nondecreasing and $Q_N(r, V_N)$ and $Q_N(s, W_N)$ are isomorphic under a permutation π_{k+1} that can be factorized into $k + 1$ transpositions, then $Q_N(r, V_N) = Q_N(s, W_N)$.

Let a be the smallest index such that $\pi_{k+1}(i) = i$; $i < a$ and $\pi_{k+1}(a) \neq a$. We can write π_{k+1} as $\tau_{k+1}\tau_k \cdots \tau_1$ such that $\tau_{k+1} = (a\ b)$ for some $b > a$ and τ_j does not move a for $j = 1, 2, \cdots, k$. This can be done by factorizing π_{k+1} into disjoint cycles and reordering the cycles such that the cycle containing a is the left-most cycle. After rotating the left-most cycle a number of times such that the last element in the cycle is a, the formula $(i_1\ i_2 \cdots i_l) = (i_l\ i_1)\ (i_{l-1}\ i_1) \cdots (i_2\ i_1)$ can be used to factorize each cycle to obtain the desired τ_j, $j = 1, 2, \cdots, N + 1$.

Denote $\tau_k\tau_{k-1} \cdots \tau_1$ by π_k. The permutation π_{k+1} can thus be written as a product of the transposition $\tau_{k+1} = (a\ b)$ and the permutation τ_k that has a factorization of k transpositions. Note that the transpositions τ_j, $j = 1, 2, \cdots, k$ do not move the replicas $1, 2, \cdots, a$ and hence $\pi_k(i) = i$ for $i = 1, 2, \cdots, a$. We can write

$$\begin{aligned} Q_N(s, W_N) &= \pi_{k+1}(Q_N(r, V_N)) \\ &= \tau_{k+1} \cdot \pi_k(Q_N(r, V_N)) \\ &= \tau_{k+1}(Q_N(r, U_N)) \end{aligned}$$

where U_N is a vote assignment such that $u_i = v_{\pi_k(i)}$. Since π_k does not move $1, 2, \cdots, a$, $u_a = v_a$ and since for all $i > a$, $v_i \geq v_a$, we have $u_b \geq u_a$. By Lemma A.3, we conclude that $Q_N(s, W_N) = Q_N(r, U_N) = \pi_k(Q_N(r, V_N))$, in other words, $Q_N(r, V_N)$ and $Q_N(s, W_N)$ are isomorphic under a permutation that can be factorized into k transpositions. By the induction hypothesis, we can conclude that $Q_N(s, W_N) = Q_N(r, V_N)$. □

APPENDIX B

Theorem 3.1: Given an availability vector P and a vote assignment V where, without loss of generality, $p_1 \leq p_2$

APPENDIX A

This Appendix describes the proofs of the lemmas used in Section II.

Lemma A.1—Removing a Replica from a Read Group: Let $V_N = (v_1, v_2, \cdots, v_N)$ be a vote assignment to N replicas. Let $G = \{g_1, g_2, \cdots, g_k\}$ be a read group in $Q_N(r, V_N)$. Then, $\forall g_i \in G: G - \{g_i\} \in Q_N(r - v_{g_i}, V_N)$.

Proof: Since $v(G) \geq r$, $v(G - \{g_i\}) \geq r - v_{g_i}$ for any $g_i \in G$. The only reason that $G - \{g_i\} \notin Q_N(r - v_{g_i}, V_N)$ is when it is not tight, i.e., there is a replica $g_k \in G$ such that $v(G - \{g_i\}) \geq r - v_{g_i} + v_{g_k}$. However, this implies that $v(G) \geq r + v_{g_k}$ which is a contradiction since G is tight.

Lemma A.2—Adding a Replica to a Read Group: Let $V_N = (v_1, v_2, \cdots, v_N)$, $V_{N+1} = (v_1, v_2, \cdots, v_{N+1})$ and let $G \in Q_N(r, V_N)$. Then:

If $v(G) \geq r + v_{N+1}$, then $G \in Q_{N+1}(r + v_{N+1}, V_{N+1})$, otherwise, $G \cup \{N+1\} \in Q_{N+1}(r + v_{N+1}, V_{N+1})$.

Proof: If $v(G) \geq r + v_{N+1}$, then G is also a tight group of $r + v_{N+1}$ votes, or $G \in Q_{N+1}(r + v_{N+1}, V_{N+1})$.

If $v(G) < r + v_{N+1}$, then G does not have a sufficient number of votes to be a group of $Q_{N+1}(r + v_{N+1}, V_{N+1})$. The group $G \cup \{N+1\}$ will have at least $r + v_{N+1}$ votes and to show that it is tight, let g_1 be the replica in G with the least number of votes. If $v_{N+1} \leq v_{g_1}$, then $N+1$ is the replica with the least number of votes in $G \cup \{N+1\}$ and removal of $N+1$ from $G \cup \{N+1\}$ leaves a group of less than $r + v_{N+1}$ votes and thus G is tight. If $v_{N+1} > v_{g_1}$, then $G \cup \{N+1\}$ satisfies

$$r + v_{N+1} \leq v(G \cup \{N+1\}) \leq (r + v_{N+1} - 1) + v_{g_1}$$

because G satisfies $r \leq v(G) \leq (r-1) + v_{g_1}$. Therefore, $G \cup \{N+1\}$ is tight and $G \cup \{N+1\} \in Q_{N+1}(r + v_{N+1}, V_{N+1})$. □

Lemma 2.1—Expanding the Vote Assignment: Let vote assignments V_N and V_{N+1} be as in Lemma A.2. Then,

$$Q_{N+1}(r, V_{N+1}) = Q_N(r, V_N) \cup \{G \cup \{N+1\} \mid G \in Q_N(r - v_{N+1}, V_N) \wedge G \notin Q_N(r, V_N)\}.$$

Proof: (By showing LHS $\subseteq$ RHS and RHS $\subseteq$ LHS.)

We first show that LHS $\subseteq$ RHS. Let $H \in Q_{N+1}(r, V_{N+1})$. If $N+1 \notin H$, then $H \in Q_N(r, V_N)$. When $N+1 \in H$, then $H \notin Q_N(r, V_N)$. Also, $H - \{N+1\} \notin Q_N(r, V_N)$ because H is a tight group of r votes, and hence $v(H - \{N+1\}) < r$. Define the group $G_1 = H - \{N+1\}$. We have, by Lemma A.1, that $G_1 \in Q_{N+1}(r - v_{N+1}, V_{N+1})$, and since $N+1 \notin G_1$, $G_1 \in Q_N(r - v_{N+1}, V_N)$. Hence, $H \in \{G_1 \cup \{N+1\} \mid G_1 \in Q_N(r - v_{N+1}, V_N) \wedge G_1 \notin Q_N(r, V_N)\}$.

We can show that RHS $\subseteq$ LHS by showing that each of the sets in the RHS are subsets of the LHS. When $H \in Q_N(r, V_N)$, it must follow that $H \in Q_{N+1}(r, V_{N+1})$. Now let $H = G_1 \cup \{N+1\}$, where $G_1 \in Q_N(r - v_{N+1}, V_N)$ and $G_1 \notin Q_N(r, V_N)$. Then, by Lemma A.2, either $G_1 \in Q_{N+1}(r, V_{N+1})$ or $G_1 \cup \{N+1\} \in Q_{N+1}(r, V_{N+1})$. The former case is not possible because $G_1 \notin Q_N(r, V_N)$ and $N+1 \notin G_1$, then $G_1 \notin Q_{N+1}(r, V_{N+1})$. Thus, $H = G_1 \cup \{N+1\} \in Q_{N+1}(r, V_{N+1})$. □

When two read coteries $Q_N(r, V_N)$ and $Q_N(s, W_N)$ are isomorphic, a permutation π maps $Q_N(r, V_N)$ to $Q_N(s, W_N)$. The replica $\pi(i)$ in $Q_N(s, W_N)$ thus plays the role of the replica i in $Q_N(r, V_N)$. The role of a replica is determined by its votes and it follows that π is dependent on the assignments V_N and W_N. The following two lemmas show that when V_N and W_N are nondecreasing, the read coteries $Q_N(r, V_N)$ and $Q_N(s, W_N)$ are isomorphic if and only if they are equal. In Lemma A.3, we consider the case when two read coteries are mapped to each other by a transposition (a transposition is a permutation that exchanges two elements) and in Lemma 2.2 we consider the general case.

Lemma A.3—Equality of Isomorphic Read Coteries under a Transposition: Let V_N and W_N be two vote assignments such that for some $a < b$, $v_a \leq v_b$, and $w_a \leq w_b$ and let τ be the transposition $(a\ b)$. Then, for some r and s, $Q_N(r, V_N)$ is isomorphic to $Q_N(s, W_N)$ under τ if and only if $Q_N(r, V_N) = Q_N(s, W_N)$.

Proof: By contradiction.

Assume that $Q_N(r, V_N)$ and $Q_N(s, W_N)$ are isomorphic under τ but not equal. It cannot be the case that $Q_N(r, V_N) \subset Q_N(s, W_N)$ or $Q_N(s, W_N) \subset Q_N(r, V_N)$ because isomorphic sets have equal number of groups. Therefore, there is a group $G \in Q_N(r, V_N)$ such that $G \notin Q_N(s, W_N)$. We split the proof into cases depending on whether a and/or b is an element of G. Write G as $G_1 \cup H$ where G_1 does not contain a and b, and $H \subseteq \{a, b\}$.

H cannot be ϕ or $\{a, b\}$, since if $G = G_1$ or $G = G_1 \cup \{a, b\}$, then $G \in Q_N(s, W_N)$ contradicting the assumption that $G \notin Q_N(s, W_N)$.

Case 1: $G = G_1 \cup \{a\}$.

If $G_1 \cup \{b\} \in Q_N(r, V_N)$ also, then $G_1 \cup \{a\} \in Q_N(s, W_N)$, contradicting that $G \notin Q_N(s, W_N)$. So it must be that $G_1 \cup \{b\} \notin Q_N(r, V_N)$ and the following chains of implications can be made from the fact that $G_1 \cup \{a\} \in Q_N(r, V_N)$ and $G_1 \cup \{b\} \notin Q_N(r, V_N)$:

$$\begin{aligned} & G_1 \cup \{a\} \in Q_N(r, V_N) \wedge \\ & \quad G_1 \cup \{b\} \notin Q_N(r, V_N) \\ \Rightarrow\ & G_1 \cup \{b\} \in Q_N(s, W_N) \wedge \\ & \quad G_1 \cup \{a\} \notin Q_N(s, W_N) \\ \Rightarrow\ & \forall g \in G_1 : v(G_1) + w_b - w_g < s \wedge v(G_1) < s \\ & \quad [G_1 \cup \{b\} \text{ is tight}] \end{aligned}$$

[13] D. Barbara and H. Garcia-Molina, "The reliability of voting mechanisms," *IEEE Trans. Comput.*, vol. C-36, no. 10, pp. 1197–1208, 1987.
[14] Z. Tong and R. Kain, "Vote assignments in weighted voting mechanisms," in *Proc. 7th Symp. Reliable Distributed Syst.*, 1988, pp. 138–143.
[15] M. Ahamad and M. Ammar, "Performance of quorum-consensus algorithms for replicated data," *IEEE Trans. Software Eng.*, vol. 15, no. 4, 1989.
[16] W. Smith and P. Decitre, "An evaluation method for analysis of the weighted voting algorithm for maintaining replicated data," in *Proc. 4th Int. Conf. Distributed Comput. Syst.*, 1984, pp. 494–502.
[17] A. F. Veinott and G. B. Dantzig, "Integral extreme points," *SIAM Rev.*, vol. 10, no. 3, pp. 371–372, 1968.
[18] J. Buzen, "Computational algorithms for closed queueing networks with exponential servers," *Commun. ACM*, vol. 16, no. 9, pp. 527–531, 1973.
[19] J. J. Rotman, *An Introduction to the Theory of Groups*. Boston: Allyn and Bacon, 1984.

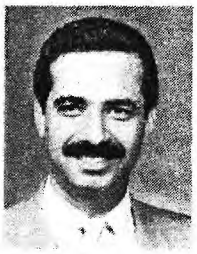
Shun Yan Cheung received the Kandidaats and Ingenieur degrees from the Department of Mathematics and Computer Science, Delft Technological University, Delft, The Netherlands, in 1981 and 1984, respectively, and the M.S. degree from the School of Information and Computer Science, Georgia Institute of Technology, Atlanta, in 1987.

He is currently a Ph.D. candidate at Georgia Tech. His current research interests include computer networks, fault tolerant systems, and performance evaluation.

Mustaque Ahamad received the B.E. (Hons.) degree in electrical engineering from the Birla Institute of Technology and Science, Pilani, India, and the M.S. and Ph.D. degrees in computer science from the State University of New York, Stony Brook, in 1983 and 1985, respectively.

Since September 1985 he has been an Assistant Professor in the School of Information and Computer Science, Georgia Institute of Technology, Atlanta. His research interests include distributed operating systems, distributed algorithms, fault-tolerant systems, and performance evaluation.

Dr. Ahamad is a member of the IEEE Computer Society.



Mostafa H. Ammar (S'83–M'85) received the S.B. and S.M. degrees from the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, in 1978 and 1980, respectively, and the Ph.D. degree from the Department of Electrical Engineering, University of Waterloo, Waterloo, Ont., Canada, in 1985.

From 1980 to 1982, he worked at Bell-Northern Research, Ottawa, Ont., Canada, first as a Member of Scientific Staff, and then as Manager of Data Network Planning. Currently, he is an Assistant Professor in the School of Information and Computer Science, Georgia Institute of Technology, Atlanta. His current research interests include information delivery systems, distributed computing and database systems, and communication networks used in manufacturing environments.

Dr. Ammar is a member of the Association for Computing Machinery, Eta Kappa Nu, and the Association of Professional Engineers of the Province of Ontario.

$\leq \cdots \leq p_N$, $v_1 \leq v_2 \leq \cdots \leq v_N$. Then,

$$\alpha(\boldsymbol{P}, \boldsymbol{V}) \geq \alpha(\boldsymbol{P}, \boldsymbol{V}')$$

for any permutation $\boldsymbol{V}'$ of the vote assignment $\boldsymbol{V}$.

Proof: We prove the theorem by showing that a vote assignment can be improved by exchanging the votes between two nodes if one of them has higher availability but is assigned less votes, that is:

Given availability vector $\boldsymbol{P}$ where, without loss of generality, $p_1 \leq p_2 \leq \cdots \leq p_N$. For the two vote assignments

$$\boldsymbol{V} = (v_1, v_2, \cdots, v_N)$$

and

$$\boldsymbol{V}' = (v_1', v_2', \cdots, v_N')$$

such that for some value l and k:

$$1 \leq l < k \leq N$$

$$v_i' = v_i \quad \text{for } i \neq k, l;$$

$$v_k' = v_l;\ v_l' = v_k \text{ and } v_l \leq v_k$$

we have

$$\alpha_s(\boldsymbol{P}, \boldsymbol{V}) \geq \alpha_s(\boldsymbol{P}, \boldsymbol{V}')$$

where s is an arbitrary threshold and $\alpha_s(\boldsymbol{P}, \boldsymbol{V})$ is given by (6).

From (6) we have

$$\alpha_s(\boldsymbol{P}, \boldsymbol{V}) = \frac{1}{K} \sum_{m=s}^{L} \sum_{\text{all } \boldsymbol{n} \text{ s.t. } C(\boldsymbol{n}, \boldsymbol{V}) = m} \prod_{i=1}^{N} q_i^{n_i}. \quad (8)$$

We define

$$h^{-lk}(\boldsymbol{V}, m) = \sum_{\text{all } \boldsymbol{n} \text{ s.t. } C^{-lk}(\boldsymbol{n}, \boldsymbol{V}) = m} \prod_{i=1, i \neq l,k}^{N} q_i^{n_i} \quad (9)$$

where $C^{-lk}(\boldsymbol{n}, \boldsymbol{V}) = \sum_{i=1, i \neq l,k}^{N} n_i v_i$. We thus have that

$$K\alpha_s(\boldsymbol{P}, \boldsymbol{V}) = \sum_{m=s}^{L} \left[h^{-lk}(\boldsymbol{V}, m) + q_l h^{-lk}(\boldsymbol{V}, m - v_l) + q_k h^{-lk}(\boldsymbol{V}, m - v_k) + q_l q_k h^{-lk}(\boldsymbol{V}, m - v_l - v_k) \right]. \quad (10)$$

Similarly, we get

$$K\alpha_s(\boldsymbol{P}, \boldsymbol{V}') = \sum_{m=s}^{L} \left[h^{-lk}(\boldsymbol{V}, m) + q_k h^{-lk}(\boldsymbol{V}, m - v_l) + q_l h^{-lk}(\boldsymbol{V}, m - v_k) + q_l q_k h^{-lk}(\boldsymbol{V}, m - v_l - v_k) \right]. \quad (11)$$

Therefore, we have

$$K(\alpha_s(\boldsymbol{P}, \boldsymbol{V}) - \alpha_s(\boldsymbol{P}, \boldsymbol{V}')) = (q_k - q_l) \sum_{m=s}^{L} \left[h^{-lk}(\boldsymbol{V}, m - v_k) - h^{-lk}(\boldsymbol{V}, m - v_l) \right]. \quad (12)$$

If $p_l \leq p_k$ and $q_i = p_i/(1 - p_i)$, we also have $q_l \leq q_k$. To prove the lemma, it is thus sufficient to show that

$$\sum_{m=s}^{L} \left[h^{-lk}(\boldsymbol{V}, m - v_k) - h^{-lk}(\boldsymbol{V}, m - v_l) \right] \geq 0. \quad (13)$$

The left-hand side of (13) can be written as

$$\sum_{m=s-v_k}^{L-v_k} h^{-lk}(\boldsymbol{V}, m) - \sum_{m=s-v_l}^{L-v_l} h^{-lk}(\boldsymbol{V}, m). \quad (14)$$

Since $v_l \leq v_k$, then also $s - v_l \geq s - v_k$ and $L - v_l \geq L - v_k$. Cancel out the identical terms in (14) and we thus get (14) equal to

$$\sum_{m=s-v_k}^{s-v_l-1} h^{-lk}(\boldsymbol{V}, m) - \sum_{m=L-v_k+1}^{L-v_l} h^{-lk}(\boldsymbol{V}, m). \quad (15)$$

Note that $h^{-lk}(\boldsymbol{V}, m) \geq 0$ for all m and $h^{-lk}(\boldsymbol{V}, m) = 0$ for $m > L - v_l - v_k$ (because when replicas j and k are removed, the system is left with $L - v_l - v_k$ votes). The right sum of (15) is equal to 0, and hence (15) is greater than or equal to 0.

References

[1] S. B. Davidson, H. Garcia-Molina, and D. Skeen, "Consistency in partitioned network," *ACM Comput. Survey*, vol. 17, no. 3, pp. 341–370, 1985.

[2] H. Gifford, "Weighted voting for replicated data," in *Proc. 7th Symp. Operat. Syst.*, 1979, pp. 150–162.

[3] D. Eager and K. Sevcik, "Achieving robustness in distributed database systems," *ACM Trans. Database Syst.*, vol. 8, no. 3, pp. 354–381, 1983.

[4] M. Herlihy, "Dynamic quorum adjustments for partitioned data," Carnegie-Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-CS-86-147, 1987.

[5] A. E. Abbadi and S. Toueg, "Maintaining availability in partitioned replicated databases," in *Proc. Symp. Principles Database Syst. (PODS)*, 1986, pp. 240–351.

[6] S. Jajodia and D. Mutchler, "Dynamic voting," in *Proc. SIGMOD-87*, 1987, pp. 227–238.

[7] D. Barbara, H. Garcia-Molina, and A. Spauster, "Protocols for dynamic vote reassignment," in *Proc. Principles of Distributed Comput.*, 1986, pp. 195–205.

[8] J. Paris, "Voting with witnesses: A consistency scheme for replicated files," in *Proc. 6th Int. Conf. Distributed Comput. Syst.*, 1986, pp. 606–612.

[9] D. Davcev and W. Burkhard, "Consistency and recovery control for replicated data," in *Proc. 10th Symp. Operat. Syst. Principles*, 1985, pp. 87–96.

[10] D. Agrawal and A. E. Abbadi, "Reducing storage for quorum consensus algorithms," in *Proc. Very Large Databases Conf.*, 1988, pp. 419–430.

[11] H. Garcia-Molina and D. Barbara, "How to assign votes in a distributed system," *J. ACM*, vol. 32, no. 4, pp. 841–860, 1985.

[12] L. Lamport, "The implementation of reliable distributed multiprocess systems," *Comput. Networks*, vol. 2, pp. 95–114, 1978.